Statistics and Computing (2021) 31:74
https://doi.org/10.1007/s11222-021-10048-0

Anytime parallel tempering

Alix Marie d'Avigneau[1] · **Sumeetpal S. Singh**[1] · **Lawrence M. Murray**[2]

Received: 25 June 2020 / Accepted: 27 August 2021

Abstract
Developing efficient MCMC algorithms is indispensable in Bayesian inference. In parallel tempering, multiple interacting MCMC chains run to more efficiently explore the state space and improve performance. The multiple chains advance independently through local moves, and the performance enhancement steps are exchange moves, where the chains pause to exchange their current sample amongst each other. To accelerate the independent local moves, they may be performed simultaneously on multiple processors. Another problem is then encountered: depending on the MCMC implementation and inference problem, local moves can take a varying and random amount of time to complete. There may also be infrastructure-induced variations, such as competing jobs on the same processors, which arises in cloud computing. Before exchanges can occur, all chains must complete the local moves they are engaged in to avoid introducing a potentially substantial bias (Proposition 1). To solve this issue of randomly varying local move completion times in multi-processor parallel tempering, we adopt the Anytime Monte Carlo framework of (Murray, L. M., Singh, S., Jacob, P. E., and Lee, A.: Anytime Monte Carlo. *arXiv preprint* arXiv:1612.03319, (2016): we impose real-time deadlines on the parallel local moves and perform exchanges at these deadlines without any processor idling. We show our methodology for exchanges at real-time deadlines does not introduce a bias and leads to significant performance enhancements over the naïve approach of idling until every processor's local moves complete. The methodology is then applied in an ABC setting, where an Anytime ABC parallel tempering algorithm is derived for the difficult task of estimating the parameters of a Lotka–Volterra predator-prey model, and similar efficiency enhancements are observed.

Keywords Bayesian inference · Markov chain Monte Carlo (MCMC) · Parallel tempering · Anytime Monte Carlo · Approximate Bayesian computation (ABC) · Likelihood-free inference

1 Introduction

Consider a set of m observations $y = \{y_1, \ldots, y_m\} \in \mathcal{Y}$ following a probability model with underlying parameters $\theta \in \Theta$ and associated *likelihood* $f(y_1, \ldots, y_m \mid \theta)$ which we abbreviate to $f(y \mid \theta)$. In most cases, the posterior $\pi(\mathrm{d}\theta)$ of interest is intractable and must be approximated using computational tools such as the commonly used *Metropolis-Hastings* (M-H) algorithm (Robert and Casella (2004)) with random walk proposals, for example. However, as models become more complex, the exploration of the posterior using such basic methods quickly becomes inefficient (Beskos et al. (2009)). Furthermore, the model itself can pose its own challenges such as the likelihood becoming increasingly costly or even impossible to evaluate (Tavaré et al. (1997)); the Lotka-Volterra predator-prey model of Sect. 5 is a concrete example.

Parallel tempering, initially proposed by Swendsen and Wang (1986) and further developed under the name Metropolis-coupled Markov chain Monte Carlo (MC)^3 by Geyer (1991), is a generic method for improving the efficiency of MCMC that can be very effective without significantly altering the original MCMC algorithm, beyond perhaps tuning its local proposals for each temperature. The parallel tempering algorithm runs multiple interacting MCMC chains to more efficiently explore the state space. The multiple MCMC chains are advanced independently, in what is known as the

✉ Alix Marie d'Avigneau
agem2@cam.ac.uk; a.marie-davigneau@soton.ac.uk

Sumeetpal S. Singh
sss40@cam.ac.uk

Lawrence M. Murray
lawrence.murray@uber.com

[1] Signal Processing and Communications Group, Department of Engineering, University of Cambridge, Cambridge, UK

[2] Uber AI Labs, San Francisco, CA, USA

Published online: 20 September 2021

local moves, and the performance enhancement steps are the exchange moves, where the chains pause and attempt to swap their current sample amongst each other. Parallel tempering allows for steps of various sizes to be made when exploring the parameter space, which makes the algorithm effective, even when the distribution we wish to sample from has multiple modes. In order to reduce the real time taken to perform the independent local moves, they may be performed simultaneously on multiple processors, a feature we will focus on in this work.

Let the parallel tempering MCMC chain be $\left(X_n^{1:\Lambda}\right)_{n=1}^{\infty} = \left(X_n^1, \ldots, X_n^\Lambda\right)_{n=1}^{\infty}$ with initial state $\left(X_0^{1:\Lambda}\right)$ and target distribution

$$\pi(\mathrm{d}x^{1:\Lambda}) \propto \prod_{\lambda=1}^{\Lambda} \pi_\lambda(\mathrm{d}x^\lambda), \tag{1}$$

where the $\pi_\lambda(\cdot)$ are independent marginals corresponding to the target distribution of each of Λ chains, running in parallel at different temperatures indexed by λ. One of these chains, say $\lambda = \Lambda$, is the *cold* chain, and its target distribution $\pi_\Lambda = \pi$ is the posterior of interest. At each step n of parallel tempering (Geyer (2011)), one of two types of updates is used to advance the Markov chain $X_n^{1:\Lambda}$ to its next state:

1. Independent *local moves*: for example, a standard Gibbs or Metropolis-Hastings update, applied to each tempered chain X_n^λ in parallel.
2. Interacting *exchange moves*: propose to swap the states $x \sim \pi_\lambda$ and $x' \sim \pi_{\lambda'}$ of one or more pairs of adjacent chains. For each pair, accept a swap with probability

$$a_{swap}(x', x) = \min\left\{1, \frac{\pi_\lambda(x')\pi_{\lambda'}(x)}{\pi_\lambda(x)\pi_{\lambda'}(x')}\right\}, \tag{2}$$

otherwise, the chains in the pair retain their current states.

With the cold chain providing the desired precision and the warmer chains more freedom of movement when exploring the parameter space, the combination of the two types of update allows all chains to mix much faster than any one of them would mix on its own. This provides a way to jump from mode to mode in far fewer steps than would be required under a standard non-tempered implementation using, say, the Metropolis-Hastings algorithm.

A particular advantage of parallel tempering is that it is possible to perform the independent local moves in parallel on multiple processors in order to reduce the real time taken to complete them. Unfortunately, this gives rise to the following problem: depending on the MCMC implementation and the inference problem itself, the local moves can take a *varying and random* amount of time to complete, which depends on the part of the state space it is exploring (see the Lotka-Volterra predator-prey model in Sect. 5.3 for a specific real example). Thus, before the exchange moves can occur, all chains *must* complete the local move they are engaged in to avoid introducing a potentially substantial bias (see Proposition 1). Additionally, the time taken to complete local moves may also reflect computing infrastructure induced variations, for example, due to variations in processor hardware, memory bandwidth, network traffic, I/O load, competing jobs on the same processors, as well as potential unforeseen interruptions, all of which affect the compute time of local moves. Local moves in parallel tempering algorithms can also have temperature-dependent completion times. This is the case of the approximate Bayesian computation (ABC) application in Sect. 4. In Earl and Deem (2004), the authors consider a similar problem of temperature λ dependent real completion times of local moves. To tackle the problem, they redistribute the chains among the processors in order to minimise processor idling that occurs while waiting for all local moves to finish. This strategy is a deterministic allocation of processor time to simulation and entails completing part of a simulation on one processor and then continuing on another. Our approach to removing idling doesn't involve redistributing partially completed simulations, and instead imposes real-time deadlines at which simulations are stopped to perform exchange moves before resuming work on their respective processors. The contributions of this paper are as follows.

Firstly, to solve the problem of randomly distributed local move completion times when parallel tempering is implemented on a multi-processor computing resource, we adopt the Anytime Monte Carlo framework of Murray et al. (2016): we guarantee the simultaneous readiness of all chains by imposing real-time deadlines on the parallelly computed local moves, and perform exchange moves at these deadlines without any idling, i.e. without waiting for the slowest of them to complete their local moves. Idling is both a financial cost, for example in a cloud computing setting, and can also significantly reduce the effective Monte Carlo sample size returned. We show that hard deadlines introduce a bias which we mitigate using the Anytime framework (see Proposition 2).

Secondly, we illustrate our gains through detailed numerical work. The first experiment considered is a mixture model where the biased and de-biased target distributions can be characterised for ease of comparison with the numerical results. We then apply our Anytime parallel tempering methodology in the realm of ABC (Tavaré et al. (1997); Pritchard et al. (1999)). In ABC, simulation is used instead of likelihood evaluations, which makes it particularly useful for Bayesian problems where the likelihood is unavailable or too costly to compute. In Lee (2012), a more efficient MCMC kernel for ABC (as measured by the effective sample size), called the *1-hit MCMC kernel*, was devised to

significantly improve the probability that a good proposal in the direction of a higher posterior density is accepted, thus more closely mimicking exact likelihood evaluations. This new MCMC kernel was subsequently shown in Lee and Łatuszyński (2014) to also theoretically outperform competing ABC methods. The 1-hit kernel has a random execution time that depends on the part of the parameter space being explored, and is thus a good candidate for our Anytime parallel tempering method. In this paper, we show that we can improve the performance of the 1-hit MCMC kernel by introducing tempering and exchange moves, and embed the resulting parallel tempering algorithm within the Anytime framework to mitigate processor idling due to random local move completion times. Parallel tempering for ABC has been proposed by Baragatti et al. (2013), but hasn't been studied in the Anytime context as we do for random local move completion times, nor has the more efficient 1-hit MCMC kernel been employed. We perform a detailed numerical study of the Lotka-Volterra predator-prey model, which has an intractable likelihood and is a popular example used to contrast methods in the ABC literature (Fearnhead and Prangle (2012); Toni et al. (2009); Prangle et al. (2017)). The time taken to simulate from the Lotka-Volterra model is random and parameter value dependent; this randomness is in addition to that induced by the 1-hit kernel.

The Anytime parallel tempering framework can be applied in several contexts. For example, another candidate for our framework is reversible jump MCMC (RJ-MCMC) by Green (1995), which is a variable-dimension Bayesian model inference algorithm. An instance of RJ-MCMC within a parallel tempering algorithm is given in Jasra et al. (2007), where multiple chains are simultaneously updating states of variable dimensions (depending on the model currently considered on each chain), and the real completion time of local moves depends on the dimension of the state space under the current model. Additionally, in the fixed dimension parallel tempering setting, if the local moves use any of the following MCMC kernels, then they have a parameter dependent completion time and thus could benefit from an Anytime formulation: the no-U-turn sampler (NUTS) (Hoffman and Gelman (2014)) and elliptical slice sampling (Murray et al. (2010); Nishihara et al. (2014)). Even if the local moves do not take a variable random time to complete by design (Friel and Pettitt (2008); Calderhead and Girolami (2009)), computer infrastructure induced variations, such as memory bandwidth, competing jobs, etc. can still affect the real completion time of local moves in a parallel tempering algorithm, such as in Rodinger et al. (2006). In the statistical mechanics literature, there are also parallel tempering-based simulation problems where the local move completion time is temperature- and parameter-dependent as well as random, e.g. see Hritz and Oostenbrink (2007); Karimi et al. (2011); Wang and Jordan (2003); Earl and Deem (2004), and thus could benefit from our Anytime formulation. Finally, the Anytime framework has not been tested beyond the SMC^2 example of Murray et al. (2016), but it can be applied to any parallelisable population-based MCMC algorithm which includes local moves and interacting moves where all processors must communicate, such as sequential Monte Carlo (SMC) samplers (Del Moral et al. (2006)), or parallelised generalised elliptical slice sampling (Nishihara et al. (2014)).

This paper is structured as follows. Sections 2 and 3 develop our Anytime Parallel Tempering Monte Carlo (APTMC) algorithm and then Sect. 4 extends our framework further for the 1-hit MCMC kernel of Lee (2012) for ABC. Experiments are run in Sect. 5 and include a carefully constructed synthetic example to demonstrate the workings and salient features of Anytime parallel tempering. Section 5 also presents an application of Anytime parallel tempering to the problem of estimating the parameters of a stochastic Lotka-Volterra predator-prey model. Finally, Sect. 6 provides a summary and some concluding remarks.

2 Anytime Monte Carlo

Let $(X_n)_{n=0}^{\infty}$ be a Markov chain with initial state X_0, evolving on state space $\mathcal{X}$, with transition kernel $X_n \,|\, x_{n-1} \sim \kappa(\mathrm{d}x_n|x_{n-1})$ and target distribution $\pi(\mathrm{d}x)$. Define the *hold time* H_{n-1} as the random and positive real time required to complete the computations necessary to transition from state X_{n-1} to X_n via the kernel κ. Then let $H_{n-1} \,|\, x_{n-1} \sim \tau(\mathrm{d}h_{n-1}|x_{n-1})$ where τ is the hold time distribution.

Assume that the hold time $H > \epsilon > 0$ for minimal time ϵ, $\sup_{x\in\mathcal{X}} \mathbb{E}[H \,|\, x] < \infty$, and the hold time distribution τ is homogeneous in time. In general, nothing is known about the hold time distribution τ except how to sample from it, i.e. by recording the time taken by the algorithm to simulate $X_n \,|\, x_{n-1}$. Let $\kappa(\mathrm{d}x_n, \mathrm{d}h_{n-1}|x_{n-1}) = \kappa(\mathrm{d}x_n|h_{n-1}, x_{n-1})\tau(\mathrm{d}h_{n-1}|x_{n-1})$ be a joint kernel. The transition kernel $\kappa(\mathrm{d}x_n|x_{n-1})$ is the marginal of the joint kernel over all possible hold times H_{n-1}. Denote by $(X_n)_{n=0}^{\infty}$ and $(H_n)_{n=0}^{\infty}$ the states and hold times of the joint process, and define the *arrival time* of the n-th state as

$$A_n := \sum_{i=0}^{n-1} H_i, \qquad n \geq 1,$$

where $a_0 := 0$. A possible realisation of the joint process is illustrated in Fig. 1.

Let the process $N(t) := \sup\{n : A_n \leq t\}$ count the number of arrivals by time t. From this, construct a continuous Markov jump process $(X, L)(t)$ where $X(t) := X_{N(t)}$ and $L(t) := t - A_{N(t)}$ is the *lag time* elapsed since the last jump. This continuous process describes the progress of the computation in real time.

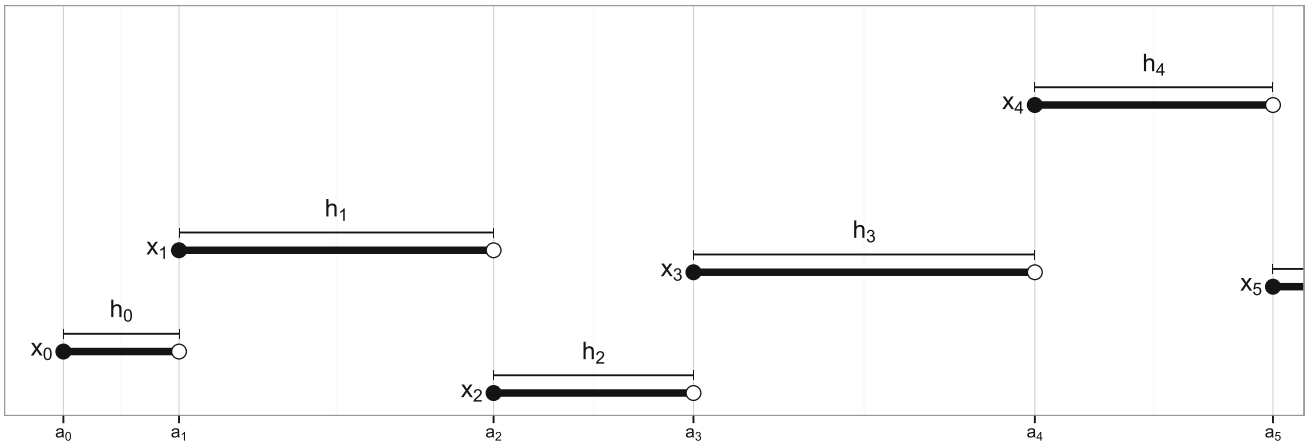

Fig. 1 (Murray et al. (2016), Fig. 1) Real-time realisation of a Markov chain with states $(X_n)_{n=0}^{\infty}$, arrival times $(A_n)_{n=0}^{\infty}$ and hold times $(H_n)_{n=0}^{\infty}$.

Proposition 1 *(Murray et al. (2016), Proposition 1) The continuous Markov jump process* $(X, L)(t)$ *has stationary distribution given by*

$$\alpha(\mathrm{d}x, \mathrm{d}l) = \frac{\bar{F}_\tau(l \mid x)}{\mathbb{E}[H]}\pi(\mathrm{d}x)\mathrm{d}l, \tag{3}$$

where $\bar{F}_\tau(l \mid x) = 1 - F_\tau(l \mid x)$, and $F_\tau(l \mid x)$ is the cumulative distribution function (cdf) of $\tau(\mathrm{d}h_n|x_n)$.

Corollary 1 *(Murray et al. (2016), Corollary 2) The marginal* $\alpha(\mathrm{d}x)$ *of the density in* (3) *is length-biased with respect to the target density* $\pi(\mathrm{d}x)$ *by expected hold time, i.e.*

$$\alpha(\mathrm{d}x) = \frac{\mathbb{E}[H \mid x]}{\mathbb{E}[H]}\pi(\mathrm{d}x). \tag{4}$$

The proofs of Proposition 1 and Corollary 1 are given in Murray et al. (2016).

The distribution α is referred to as the *anytime distribution* and is the stationary distribution of the Markov jump process. Note that Proposition 1 suggests that when the real time taken to draw a sample depends on the state of the Markov chain, i.e. $\mathbb{E}[H \mid x] \neq \mathbb{E}[H]$, a length bias with respect to computation time is introduced. In other words, when interrupted at real time t, the state of a Monte Carlo computation targeting π is distributed according to the anytime distribution α, which can essentially be seen as a length-biased target distribution. This bias diminishes with time, and when an empirical approximation or average over all post burn-in samples is required, it may be rendered negligible for a long enough computation. However, the bias in the final state does not diminish with time, and when this final state is important—which is the case in parallel tempering—the bias cannot be avoided by running the algorithm for longer. We now discuss the approach in Murray et al. (2016) to correct this bias. The main idea is to make it so expected hold time is independent of X, which leads to $\mathbb{E}[H \mid x] = \mathbb{E}[H]$ and hence $\alpha(\mathrm{d}x) = \pi(\mathrm{d}x)$, following Corollary 1. This is trivially the case for iid sampling as $\kappa(\mathrm{d}x|x_{n-1}) = \pi(\mathrm{d}x)$, so the hold time H_{n-1} for X_{n-1} is the time taken to sample $X_n \sim \pi(\mathrm{d}x)$, and therefore independent of the state X_{n-1}. One approach to non-iid sampling involves simulating $K+1$ Markov chains for $K > 0$, where we assume for now that all the Markov chains are targeting π and using the same transition kernel κ and hold time distribution τ. These $K+1$ chains are simulated on the same processor in a serial schedule. This ensures that whenever the real-time deadline t is reached, states from all but one of the chains, say the $(K+1)$-th chain, are independently distributed according to the target π. Since the $(K+1)$-th chain is the currently working chain, i.e. the latest to go through the simulation process, its state at the real-time deadline is distributed according to the anytime distribution α. Simply discarding or ignoring the state of this $(K+1)$-th chain eliminates the length bias. See Murray et al. (2016) (Section 2.1) for more details.

Using this multiple chain construction, it is thus possible draw samples from π by interrupting the process at any time t. This sets the basis for the focus of this paper: the Anytime Parallel Tempering Monte Carlo (APTMC) algorithm, described next. From this point onward, the number of chains on a given worker or processor within the Anytime framework is referred to as K rather than $K+1$ for simplicity.

3 Anytime parallel tempering Monte Carlo (APTMC)

3.1 Overview

Consider the problem in which we wish to sample from target distribution $\pi(\mathrm{d}x)$. In a parallel tempering framework, construct Λ Markov chains where each individual chain λ targets the tempered distribution

$$\pi_\lambda(\mathrm{d}x) \propto \pi(\mathrm{d}x)^{\frac{\lambda}{\Lambda}}$$

and is associated with kernel $\kappa_\lambda(\mathrm{d}x_n \mid \mathrm{d}x_{n-1})$ and hold time distribution $\tau_\lambda(\mathrm{d}h_n \mid x_n)$. In this setting, the hold time distribution is not assumed to be homogeneous across all chains, and may be temperature-dependent. Assume that all Λ chains are running concurrently on Λ processors. We aim to interrupt the computations on a real-time schedule of times $t_1, t_2, t_3, \ldots$ to perform exchange moves between adjacent pairs of chains before resuming the local moves. To illustrate the challenge of this task, we discuss the case where $\Lambda = 2$. Let π_2 be the desired posterior and π_1 the 'warm' chain, with associated hold time distributions τ_1 and τ_2, respectively. When the two chains are interrupted at some time t, assume that the current sample on chain 1 is X_m^1 and that of chain 2 is X_n^2. It follows from Corollary 1 that

$$X_m^1 \sim \alpha_1(\mathrm{d}x) = \frac{\mathbb{E}[H_1 \mid x]}{\mathbb{E}[H_1]}\pi_1(\mathrm{d}x) \neq \pi_1(\mathrm{d}x),$$

and similarly for X_n^2. Exchanging the samples using the acceptance probability in (2) is incorrect. Indeed, exchanging using the current samples X_m^1 and X_n^2, if accepted, will result in the sample sets $\{X_1^1, X_2^1, \ldots\}$ and $\{X_1^2, X_2^2, \ldots\}$ being corrupted with samples which arise from their respective length-biased, anytime distributions α_1 and α_2, as opposed to being exclusively from π_1 and π_2. Furthermore, the expressions for α_1 and α_2 will most often be unavailable, since their respective hold time distributions τ_1 and τ_2 are not explicitly known but merely implied by the algorithm used to simulate the two chains. Finally, we could wait for chains 1 and 2 complete their computation of X_{m+1}^1 and X_{n+1}^2 respectively, and then accept/reject the exchange $(X_{m+1}^1, X_{n+1}^2) \to (X_{n+1}^2, X_{m+1}^1)$ according to (2). This approach won't introduce a bias but can result in one processor idling while the slower computation finishes. We show this can result in significant idling in numerical examples.

In the next section, we describe how to correctly implement exchange moves within the Anytime framework.

3.2 Anytime exchange moves

Here, we adapt the multi-chain construction devised to remove the bias present when sampling from Λ Markov chains, where each chain λ targets the distribution π_λ for $\lambda = 1, \ldots, \Lambda$. Associated with each chain is MCMC kernel $\kappa_\lambda(\mathrm{d}x_n^\lambda \mid \mathrm{d}x_{n-1}^\lambda)$ and hold time distribution $\tau_\lambda(\mathrm{d}h \mid x)$.

Proposition 2 *Let $\pi_\lambda(\mathrm{d}x)$, $\lambda = 1 \ldots, \Lambda$ be the stationary distributions of Λ Markov chains with associated MCMC kernels $\kappa_\lambda(\mathrm{d}x_n^\lambda \mid \mathrm{d}x_{n-1}^\lambda)$ and hold time distributions $\tau_\lambda(\mathrm{d}h \mid x)$. Assume the chains are updated sequentially and let j be the index of the currently working chain. The joint anytime distribution is the following generalisation of Proposition 1*

$$A(\mathrm{d}x^{1:\Lambda}, \mathrm{d}l, j) = \frac{1}{\Lambda}\frac{\mathbb{E}[H \mid j]}{\mathbb{E}[H]}\alpha_j(\mathrm{d}x^j, \mathrm{d}l) \prod_{\lambda=1,\, \lambda\neq j}^{\Lambda} \pi_\lambda(\mathrm{d}x^\lambda).$$

The proof of Proposition 2 is given in Appendix A.1. Conditioning on x^j, j and l we obtain

$$A(\mathrm{d}x^{1:\Lambda\setminus j} \mid x^j, l, j) = \prod_{\lambda=1,\, \lambda\neq j}^{\Lambda} \pi_\lambda(\mathrm{d}x^\lambda). \tag{5}$$

Therefore, if exchange moves on the conditional $A(\mathrm{d}x^{1:\Lambda\setminus j} \mid x^j, l, j)$ are performed by 'eliminating' the j-th chain to obtain the expression in (5), they are being performed involving only chains distributed according to their respective targets π_λ and thus the bias is eliminated.

3.3 Implementation

On a single processor, the algorithm may proceed as in Algorithm 1, where in Step 3 the Λ chains are simulated one at a time in a serial schedule. Figure 2 provides an illustration of how the algorithm works.

Algorithm 1 Anytime Parallel Tempering Monte Carlo on one processor (`APTMC-1`)

1: Initialise real-time Markov jump process $(X^{1:\Lambda}, L, J)(0) = (x_0^{1:\Lambda}, 0, 1)$.
2: Set $n^{1:\Lambda} := 0$. ▷ *number of samples per chain*
3: **for** $i = 1, 2, \ldots$ **do**
SIMULATE REAL-TIME MARKOV JUMP PROCESS $(X^{1:\Lambda}, L, J)(t)$ UNTIL REAL TIME t_i.
4: Perform local moves on $x_{n^j}^j$.
5: $j := j + 1 \mod \Lambda$
6: $n^j := n^j + 1$
PERFORM EXCHANGE STEPS ON THE CONDITIONAL IN (5).
7: Select one or more pair(s) of adjacent chains with indices taken from the set $\{1 : \Lambda\} \setminus j$.
8: Propose to swap the selected pair(s) of states $(x_{n^\lambda}^\lambda, x_{n^{\lambda'}}^{\lambda'})$ according to Algorithm 2.
9: **end for**

When multiple processors are available, the Λ chains can be allocated to them. However, running a single chain on each processor means that when the real-time deadline occurs, all chains will be distributed according to their respective anytime distributions α_λ, and thus be biased as exchange moves occur. Therefore, all processors must contain at least two chains. A typical scenario would be each processor is allocated two or more temperatures to sample from. The implementation is defined as described in Algorithm 3. Note that the multiple chain construction eliminates the intractable

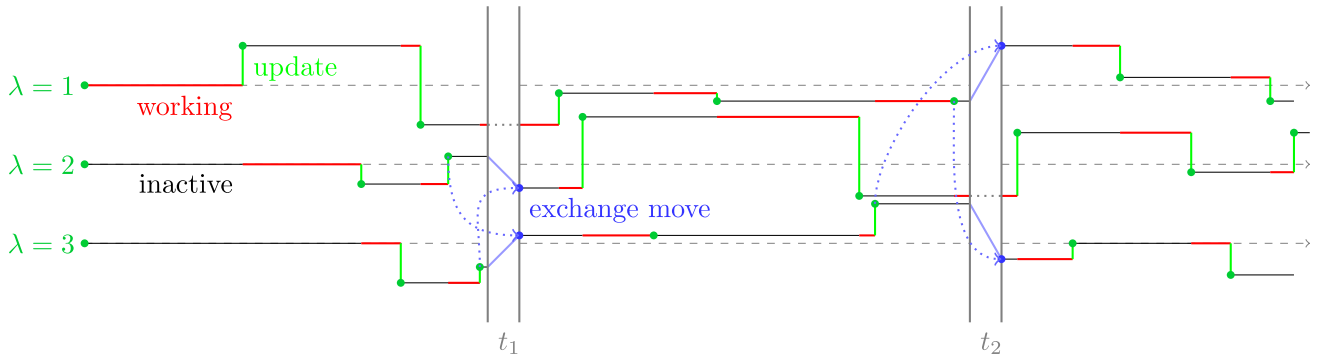

Fig. 2 Illustration of the progression of three chains in the APTMC algorithm on a single processor. The *green* (local move) and *blue* (exchange move) dots represent samples from the posterior being recorded as their respective local and exchange moves are completed. When exchange moves occur at t_1, chain $\lambda = 1$ is currently moving and cannot participate in exchange moves without introducing a bias. Therefore it is ignored, and the exchange moves are performed on the remaining (inactive) chains. Similarly, at time t_2 chain $\lambda = 2$ is excluded from the exchange. The widths of intervals t_1 and t_2 are for illustrating the exchange procedure only. (Color figure online)

Algorithm 2 Exchange move between two chains

Input: states $(x_n^{\lambda}, x_{n'}^{\lambda'})$ where $x_n^{\lambda} \sim \pi_{\lambda}$ and $x_{n'}^{\lambda'} \sim \pi_{\lambda'}$.

Compute exchange move acceptance probability $a_{swap}(x_n^{\lambda}, x_{n'}^{\lambda'})$ given in (2).
Sample $u \sim \text{Uniform}(0, 1)$.
if $u < a_{swap}(x_n^{\lambda}, x_{n'}^{\lambda'})$ **then**
    $(x_{n+1}^{\lambda}, x_{n'+1}^{\lambda'}) = (x_{n'}^{\lambda'}, x_n^{\lambda})$
else
    $(x_{n+1}^{\lambda}, x_{n'+1}^{\lambda'}) = (x_n^{\lambda}, x_{n'}^{\lambda'})$
end if
$n := n + 1$ and $n' := n' + 1$.

Output: updated states $(x_{n+1}^{\lambda}, x_{n'+1}^{\lambda'})$.

densities in the acceptance ratio for the exchange step when τ differs between processors, since exchange moves are performed between chains that are not currently working (i.e. on density (5) for a single processor and (6) for multiple processors), so the hold time distribution does not factor in.

Depending on the problem at hand and computing resources available, there are various approaches to distributing the chains across workers. We distinguish three possible scenarios. The first is an ideal scenario, where the number of processors exceeds Λ and the communication overhead between workers is negligible. In this scenario, each worker implements $K = 2$ chains running at the same temperature. For example, with $W = \Lambda$ workers, worker $w = \lambda$ contains 2 chains targeting π_{λ}. The second scenario arises when the number of workers available is limited, but communication overhead is still negligible. In this case, the chains, sorted in increasing order of temperature, are divided evenly among workers. For example, with $W = \frac{\Lambda}{2}$ workers, worker w could contain two chains, one with target π_{2w-1} and one with target π_{2w}. The third scenario deals with non-negligible inter-processor communication overhead (which only affects the exchange moves). To account for this, exchange moves are divided into two types:

1. *Within-worker* exchange move: performed on each individual worker in parallel, between a pair of adjacent chains. No communication between workers is necessary in this case.
2. *Between-worker* exchange move: performed by selecting a pair of adjacent workers and exchanging between the warmest eligible chain from the first worker and coldest from the second. Thus, an exchange move between two adjacent chains is effectively being performed, except this time communication between workers is required.

3.4 Tuning considerations

In this section we discuss the issue of tuning Anytime parallel tempering by drawing on various ideas from the literature. The main concerns are the selection of the number of chains and their temperatures, the tuning of the local moves for each chain and the selection of appropriate hard deadlines for the exchange moves to occur. In our setting, the computational budget determines the number of chains Λ, and for such a fixed budget we aim to improve sampling of the cold chain through the adoption of parallel tempering stages. The issue of determining the temperature of adjacent chains has been considered in Rathore et al. (2005); Kone and Kofke (2005); Atchadé et al. (2011) where it was shown that an exchange success rate of approximately 20-25% for adjacent chains is optimal, in an appropriate sense, and is demonstrated to confer the most benefit to sampling the coldest chain. However, the optimality curve (Kone and Kofke (2005); Atchadé et al. (2011)) has a broad mode, and even 40% seems appropriate. To achieve this 25% acceptance rate of exchange moves,

Algorithm 3 Anytime Parallel Tempering Monte Carlo on multiple processors (APTMC-W)

1: On worker w, initialise the real-time Markov jump process $\left(X_w^{1:K}, L_w, J_w\right)(0) = \left(x_{w,0}^{1:K}, 0, 1\right)$.
2: Set $n_w^{1:K} := 0$. ▷ *number of samples per chain*
3: **for** $i = 1, 2, \ldots$ **do**
ON EACH WORKER w, SIMULATE THE REAL-TIME MARKOV JUMP PROCESS $\left(X_w^{1:K}, L_w, J_w\right)(t)$ UNTIL REAL TIME t_i.
4: Perform local moves on $x_{w,n_w^{j_w}}^{j_w}$.
5: $j_w := j_w + 1 \mod K$
6: $n_w^{j_w} := n_w^{j_w} + 1$
ACROSS ALL WORKERS, PERFORM EXCHANGE STEPS ON THE CONDITIONAL

$$A(\mathbf{d}\boldsymbol{x}^{1:K\setminus \boldsymbol{j}} \mid \boldsymbol{x}^{\boldsymbol{j}}, \boldsymbol{l}, \boldsymbol{j}) = \prod_{w=1}^{W} \prod_{k=1,\, k\neq j_w}^{K} \pi_w(\mathrm{d}x_w^k), \tag{6}$$

WHERE $\mathbf{d}\boldsymbol{x}^{1:K\setminus \boldsymbol{j}} = \left(\mathrm{d}x_1^{1:K\setminus j_1}, \ldots, \mathrm{d}x_W^{1:K\setminus j_W}\right)$, $\boldsymbol{x}^{\boldsymbol{j}} = \left(x_1^{j_1}, \ldots, x_W^{j_W}\right)$, $\boldsymbol{l} = l_{1:W}$ AND $\boldsymbol{j} = j_{1:W}$.
7: For the exchange moves, combine all chains by relabelling the state indices as follows:

$$z_{m^{l(w,k)}}^{l(w,k)} = x_{w,n_w^{l(w,k)}}^{l(w,k)},$$

where $l(w,k) = (w-1)K + l$ for $k = 1, \ldots K$ and $w = 1, \ldots, W$.
8: Select one or more pair(s) of adjacent chains with indices taken from the set $\{1:\Lambda\} \setminus l(1:W, \boldsymbol{j})$.
9: Propose to swap the selected pair(s) of states $(z_{m^{\lambda}}^{\lambda}, z_{m^{\lambda'}}^{\lambda'})$ according to Algorithm 2.
10: **end for**

other than employing pilot runs, adaptive tuning is possible and Miasojedow et al. (2013) use a Robbins-Munro scheme to adjust the temperatures to target a 25% acceptance rate during runtime. The next issue is local proposals, and how large a change of state one should attempt (for the local accept/reject step). This subject has received ample attention in the literature following the seminal paper by Roberts et al. (2001), where a 25% local move acceptance rate is again optimal. The local proposal can be a Gaussian proposal whose mean and covariance matrix are tuned online (Miasojedow et al. (2013)) via a Robbins-Munro scheme to achieve the 25% local move acceptance rate. The tuning of Gaussian proposals for MCMC in general was popularised by the seminal paper of Haario et al. (2001).

When performing exchange moves, rather than selecting a single pair of adjacent chains from $\{(1,2), (2,3), \ldots, (\Lambda-1, \Lambda)\}$ for an exchange, it is common to propose to swap multiple pairs of chains simultaneously, as the exchange move is relatively cheap. To avoid selecting the same chain twice, they are divided into odd $\{(1,2), (3,4), \ldots\}$ and even $\{(2,3), (4,5), \ldots\}$ pairs of indices in Lingenheil et al. (2009), and all odd or even pairs are selected for exchange with equal probability. It is however shown in Syed et al. (2019) that it is better to deterministically cycle between exchanging odd and even pairs.

Although thus far we have suggested tuning the number of chains and annealing schedule for APTMC as if one were tuning a standard parallel tempering algorithm, there are some caveats which we now highlight. Selecting chains for exchange moves can be applied by omitting the currently working chains and relabelling the indices of the remaining, inactive or *eligible* chains. However, note that by the nature of the Anytime exchange moves, the Anytime version of an optimised parallel tempering algorithm can be suboptimal, since one or more temperature(s) might be missing from exchange moves. Considering the example in Fig. 2 and assuming the chains are all running at increasing temperatures, at t_2, chain 2 is working, so the exchange move is performed between chains 1 and 3. In a practical example, these chains would be further apart, which would lead to a lower exchange move acceptance rate. Selecting adjacent chains to target a slightly higher successful exchange rate, say 40%, would mitigate this issue; noting that even 40% is close to optimal Kone and Kofke (2005); Atchadé et al. (2011). In our implementation, we only experienced a small drop in acceptance rate due to attempting to swap two eligible chains that are not immediately adjacent, and this event becomes less likely as the number of chains increases.

Another important facet of tuning APTMC is the issue of determining the real-time schedule $t_1, t_2, \ldots$ of exchange moves. Let δ be the real-time interval or deadline between exchange moves, so that $t_i = i\delta$ for $i = 1, 2, \ldots$ We now present guidelines for calibrating δ. Let K be the number of chains, labelled $k = 1, \ldots, K$, on the slowest processor w_s (generally the one containing the cold chain), our experiments have shown that exchange moves should occur once every chain on this processor has completed at least one local move (see Dupuis et al. (2012) for an alternative view advocating an *infinite* exchange frequency version of parallel tempering). The expected hold time of one set of local moves on processor w_s, denoted $\mathsf{H} := \sum_{k=1}^{K} \mathbb{E}[H_k]$, can be estimated by repeatedly measuring the time taken for one set of local moves to complete, and averaging across all measurements. Using a pilot run, an estimate $\widehat{\mathsf{H}}$ of this expected hold time can be obtained, then set $\delta = \widehat{\mathsf{H}}$ for running the APTMC algorithm. This δ value can also be calibrated in real time, denoted $\delta(t)$ where t is the real time. At $t = 0$, initialise $\delta(0) = \delta_0$ such that $\delta_0 > 0$ is an initial, user-defined guess. Similarly as before, record a hold time sample every time a set of local move occurs on processor w_s, then after every exchange move, recompute $\widehat{\mathsf{H}}$ and update $\delta(t) = \widehat{\mathsf{H}}$. An advantage of this second approach is that $\delta(t)$ then adapts to a potentially time-inhomogeneous hold time, due e.g. to competing jobs on the processors starting mid-algorithm and suddenly slowing down the computation time of local moves.

A scenario we encountered in our experiments was non-negligible communication overhead between workers to execute the exchange moves, and this overhead was comparable to local move times which were themselves lengthy. To mitigate the communication overhead, as described in Sect. 3.3, exchange moves are divided into within- and between- workers. On a given worker with K chains, a set of worker specific moves is performed before inter-worker exchanges. These were K local moves, one (set of) within-worker exchange moves, then K more local moves, before inter-worker communication occurs for between-worker exchange moves. Given that within-worker exchanges are instant, this amounts in real time to performing $2K$ local moves on this worker before inter-worker communication occurs. Therefore, the real-time deadline in the Anytime version of the algorithm for this scenario is set to be $\delta = 2\mathsf{H}$ and can be determined as above. See Sect. 5.3.2 for an example.

Finally, Sect. 5.1.3 details other, empirical tools that help with tuning by assessing the efficiency of each chain. These include evaluating the sample autocorrelation function (acf), as well as the integrated autocorrelation time (IAT) and effective sample size (ESS).

4 Application to approximate Bayesian computation (ABC)

In this section we adapt the APTMC framework to ABC.

4.1 Overview of ABC

The notion of ABC was developed by Tavaré et al. (1997) and Pritchard et al. (1999). It can be seen as a likelihood-free way to perform Bayesian inference, using instead simulations from the model or system of interest, and comparing them to the observations available.

Let $y \in \mathbb{R}^d$ be some data with underlying unknown parameters $\theta \sim p(\mathrm{d}\theta)$, where $p(\theta)$ denotes the prior for $\theta \in \Theta$. Suppose we are in the situation in which the likelihood $f(y \mid \theta)$ is either intractable or too computationally expensive, which means that MCMC cannot be performed as normal. Assuming that it is possible to sample from the density $f(\cdot \mid \theta)$ for all $\theta \in \Theta$, approximate the likelihood by introducing an artificial likelihood f^ε of the form

$$f^\varepsilon(y \mid \theta) = \mathrm{Vol}(\varepsilon)^{-1} \int_{B_\varepsilon(y)} f(x \mid \theta)\mathrm{d}x, \tag{7}$$

where $B_\varepsilon(y)$ denotes a metric ball centred at y of radius $\varepsilon > 0$ and $\mathrm{Vol}(\varepsilon)$ is its volume. The resulting approximate posterior is given by

$$p^\varepsilon(\theta \mid y) = \frac{p(\theta) f^\varepsilon(y \mid \theta)}{\int p(\vartheta) f^\varepsilon(y \mid \vartheta)\mathrm{d}\vartheta}.$$

The likelihood $f^\varepsilon(y \mid \theta)$ cannot be evaluated either, but an MCMC kernel can be constructed to obtain samples from the approximate posterior $\pi^\varepsilon(\theta, x)$ defined as

$$\begin{aligned}\pi^\varepsilon(\theta, x) &= p^\varepsilon(\theta, x \mid y) \\ &\propto p(\theta) f(x|\theta) \mathbb{1}_\varepsilon(x) \mathrm{Vol}(\varepsilon)^{-1},\end{aligned}$$

where $\mathbb{1}_\varepsilon(x)$ is the indicator function for $x \in B_\varepsilon(y)$. This is referred to as *hitting* the ball $B_\varepsilon(y)$. In the MCMC kernel, one can propose $\theta' \sim q(\mathrm{d}\theta' \mid \theta)$ for some proposal density q, simulate the dataset $x \sim f(\mathrm{d}x \mid \theta')$ and accept θ' as a sample from the posterior if $x \in B_\varepsilon(y)$.

The *1-hit MCMC kernel*, proposed by Lee (2012) and described in Algorithm 4 introduces local moves in the form of a 'race': given current and proposed parameters θ and θ', respectively simulate corresponding datasets x and x' sequentially. The state associated with the first dataset to hit the ball $B_\varepsilon(y)$ 'wins' and is accepted as the next sample in the Markov chain. The proposal θ' is also accepted if both x and x' hit the ball at the same time.

Algorithm 4 1-hit MCMC kernel for ABC

Input: current state (θ_n, x_n).
Propose $\theta' \sim q(\mathrm{d}\theta \mid \theta_n)$. ▷ *propose a local move*
Compute preliminary acceptance probability. ▷ *prior check*

$$a(\theta_n, \theta') = \min\left\{1, \frac{p(\theta')q(\theta_n \mid \theta')}{p(\theta_n)q(\theta' \mid \theta_n)}\right\}.$$

Sample $u \sim \mathrm{Uniform}(0, 1)$.
if $u < a(\theta_n, \theta')$ **then**
  RACE := TRUE
else
  RACE := FALSE
  Retain $(\theta_{n+1}, x_{n+1}) = (\theta_n, x_n)$. ▷ *reject θ' as it is unlikely to win race*
end if
while RACE **do**
  Simulate $x \sim f(\mathrm{d}x \mid \theta_n)$ and $x' \sim f(\mathrm{d}x' \mid \theta')$.
  if $x \in B_\varepsilon(y)$ **or** $x' \in B_\varepsilon(y)$ **then** ▷ *stop the race once either x or x' hits the ball*
    RACE := FALSE
  end if
end while
if $x' \in B_\varepsilon(y)$ **then** ▷ *accept or reject move*
  Set $(\theta_{n+1}, x_{n+1}) = (\theta', x')$.
else
  Retain $(\theta_{n+1}, x_{n+1}) = (\theta_n, x)$.
end if
Output updated state (θ_{n+1}, x_{n+1}).

4.2 Anytime parallel tempering Monte Carlo for approximate Bayesian computation (ABC-APTMC)

Including the 1-hit kernel in the local moves of a parallel tempering algorithm is straightforward. Exchange moves must however be adapted to this new likelihood-free setting. Additionally, the race that occurs takes a random real time to complete, and this time is temperature-dependent, as it is quicker to hit a ball of larger radius. This provides good motivation for the use of Anytime Monte Carlo.

4.2.1 Exchange moves

The exchange moves for ABC are derived similarly as in Baragatti et al. (2013). Let (θ, x) and (θ', x') be the states of two chains targeting π^{ε} and $\pi^{\varepsilon'}$, respectively, where $\varepsilon' > \varepsilon$. Here, this is equivalent to saying θ' is the state of the 'warmer' chain. We already know that x' falls within ε' of the observations y, i.e. $x' \in B_{\varepsilon'}(y)$. Similarly, we also know that $x \in B_{\varepsilon}(y)$, and trivially that $x \in B_{\varepsilon'}(y)$. If x' also falls within ε of y, then swap the states, otherwise do not swap. The odds ratio is

$$\begin{aligned}&\frac{\pi^{\varepsilon'}(\theta, x)\pi^{\varepsilon}(\theta', x')}{\pi^{\varepsilon}(\theta, x)\pi^{\varepsilon'}(\theta', x')}\\ &= \frac{p(\theta)f(x \mid \theta)\mathrm{Vol}(\varepsilon')p(\theta')f(x' \mid \theta')\mathbb{1}_{\varepsilon}(x')\mathrm{Vol}(\varepsilon)}{p(\theta)f(x \mid \theta)\mathrm{Vol}(\varepsilon)p(\theta')f(x' \mid \theta')\mathrm{Vol}(\varepsilon')}\\ &= \mathbb{1}_{\varepsilon}(x'),\end{aligned}$$

so the probability of the swap being accepted is the probability of x' also hitting the ball of radius ε centred at y. This type of exchange move is summarised in Algorithm 5.

Algorithm 5 ABC: exchange move between two chains

Input: states $\omega_n = \big((\theta, x), (\theta', x')\big)$ where $\theta \sim \pi$, $x \sim f(\mathrm{d}x \mid \theta)$ and $\theta' \sim \pi'$, $x' \sim f(\mathrm{d}x' \mid \theta')$.
▷ *both* (θ, x) *and* (θ', x') *are outputs from Algorithm 4 for different* $\varepsilon' > \varepsilon$
1: **if** $x' \in B_{\varepsilon}(y)$ **then** ▷ *accept or reject swap depending on whether* x' *also hits the ball of radius* ε
2: Set $\omega_{n+1} = \big((\theta', x'), (\theta, x)\big)$.
3: **else**
4: Retain $\omega_{n+1} = \omega_n$.
5: **end if**
6: $n := n + 1$
Output: updated states ω_{n+1}.

4.2.2 Implementation

The full implementation of the ABC-APTMC algorithm on a single processor (`ABC-APTMC-1`) is described in Algorithm 6. The multi-processor algorithm can similarly be modified to reflect these new exchange moves and the resulting algorithm is referred to as `ABC-APTMC-W`.

Algorithm 6 ABC: Anytime Parallel Tempering Monte Carlo Algorithm (`ABC-APTMC-1`)

1: Initialise the real-time Markov jump process $(\theta^{1:\Lambda}, L, J) = (\theta_0^{1:\Lambda}, 0, 1)$.
2: Set $n := 0$.
3: **for** $i := 1, 2, \ldots$ **do**
SIMULATE THE REAL-TIME MARKOV JUMP PROCESS $(\theta, L, J)(t)$ UNTIL REAL TIME t_i.
4: Perform local moves on $\left(\theta_n^j, x_n^j\right)$ according to Algorithm 4.
5: $j := j + 1 \mod \Lambda$
PERFORM EXCHANGE STEPS ON THE CONDITIONAL:

$$A(\mathrm{d}\theta^{1:\Lambda} \mid \theta^j, l, j) = \prod_{\lambda=1, \lambda \neq j}^{\Lambda} \pi_{\lambda}(\mathrm{d}\theta^{\lambda}).$$

6: Perform exchange moves on $\omega_n = \left((\theta_n^{\lambda}, x_n^{\lambda}), (\theta_n^{\lambda'}, x_n^{\lambda'})\right)$ according to Algorithm 5.
7: **end for**

5 Experiments

In this section, we first illustrate the workings of the algorithms presented in Sect. 3.3 on a simple model, in which real-time behaviour is simulated using virtual time and an artificial hold distribution. The model is also employed to demonstrate the gain in efficiency provided by the inclusion of exchange moves. Then, the ABC version of the algorithms, as presented in Sect. 4, is applied to two case studies. The first case is a simple model and serves to verify the workings of the ABC algorithm, including bias correction. The second case considers the problem of estimating the parameters of a stochastic Lotka-Volterra predator-prey model — in which the likelihood is unavailable — and serves to evaluate the performance of the Anytime parallel tempering version of the ABC-MCMC algorithm, as opposed to the standard versions (with and without exchange moves) on both a single and multiple processors. The exchange moves are set up so that multiple pairs could be swapped at each iteration. All experiments in this paper were run on MATLAB and the code is available at https://github.com/alixma/ABCAPTMC.git.

5.1 Analytic example: Gamma mixture model

In this example we attempt to sample from an equal mixture of two Gamma distributions using the APTMC algorithm. Define the target $\pi(\mathrm{d}x)$ and an 'artificial' hold time $\tau(\mathrm{d}h \mid x)$ distribution as follows:

$$X \sim \phi\,\text{Gamma}(k_1, \theta_1) + (1-\phi)\,\text{Gamma}(k_2, \theta_2),$$
$$H \mid x \sim \psi\,\text{Gamma}\left(\frac{x^p}{\theta_1}, \theta_1\right) + (1-\psi)\,\text{Gamma}\left(\frac{x^p}{\theta_2}, \theta_2\right),$$

with mixture coefficients $\phi = \frac{1}{2}$ and ψ, where $\text{Gamma}(\cdot, \cdot)$ denotes the probability density function of a Gamma distribution, with shape and scale parameters (k_1, θ_1) and (k_2, θ_2) for each components, respectively, and with polynomial degree p, assuming it remains constant for both components of the mixture.

In the vast majority of experiments, the explicit form of the hold time distribution τ is not known, but observed in the form of the time taken by the algorithm to simulate X. For this example, so as to avoid external factors such as competing jobs affecting the hold time, we assume an explicit form for τ is known and simulate virtual hold times. This consists of simulating a hold time $h \sim \tau(\mathrm{d}h \mid x)$ and advancing the algorithm forward for h units of virtual time without updating the chains, effectively 'pausing' the algorithm. These virtual hold times are introduced such that what in a real-time example would be the effects of constant $(p = 0)$, linear $(p = 1)$, quadratic $(p = 2)$ and cubic $(p = 3)$ computational complexity can be studied. Another advantage is that the anytime distribution $\alpha_\Lambda(\mathrm{d}x)$ of the cold chain can be computed analytically and is the following mixture of two Gamma distributions

$$\alpha_\Lambda(\mathrm{d}x) = \varphi(p, k_{1:2}, \theta_{1:2})\,\text{Gamma}\,(k_1 + p, \theta_1) + [1 - \varphi(p, k_{1:2}, \theta_{1:2})]\,\text{Gamma}\,(k_2 + p, \theta_2)\,, \quad (8)$$

where

$$\varphi(p, k_{1:2}, \theta_{1:2}) = \left(1 + \frac{\Gamma(k_1)\Gamma(p + k_2)\theta_2^p}{\Gamma(k_2)\Gamma(p + k_1)\theta_1^p}\right)^{-1}.$$

We refer the reader to Appendix A.2 for the proof of (8). In the anytime distribution, one of the components of the Gamma distribution will have an associated mixture coefficient $\varphi(p, k_{1:2}, \theta_{1:2})$ or $1 - \varphi(p, k_{1:2}, \theta_{1:2})$ which increases with p while the coefficient of the other component decreases proportionally. Note that for constant $(p = 0)$ computational complexity, the anytime distribution is equal to the target distribution π.

5.1.1 Implementation

On a single processor, the Anytime Parallel Tempering Monte Carlo algorithm (referred to as `APTMC-1`) is implemented as follows: simulate $\Lambda = 8$ Markov chains, each targeting the distribution $\pi_\lambda(\mathrm{d}x) = \pi(\mathrm{d}x)^{\frac{\lambda}{\Lambda}}$. To construct a Markov chain $(X^\lambda)_{n=0}^\infty$ with target distribution

$$\pi_\lambda(x) \propto \left[\frac{1}{2}\text{Gamma}\,(k_1, \theta_1) + \frac{1}{2}\,\text{Gamma}\,(k_2, \theta_2)\right]^{\frac{\lambda}{\Lambda}}$$

for $\lambda = 1, \ldots, \Lambda$, use a *Random Walk Metropolis* update, i.e. symmetric Gaussian proposal distribution $\mathcal{N}(x_n^\lambda, \sigma^2)$ with mean x_n^λ and standard deviation $\sigma = 0.5$. Set $(k_1, k_2) = (3, 20)$, $(\theta_1, \theta_2) = (0.15, 0.25)$ and use $p \in \{0, 1, 2, 3\}$. The single processor algorithm is run for $T = 10^8$ units of virtual time, with exchange moves alternating between occurring on all odd $(1, 2), (3, 4), (5, 6)$ and all even $(2, 3), (4, 5), (6, 7)$ pairs of inactive chains every $\delta = 5$ units of virtual time. When the algorithm is running, a sample is recorded every time a local or exchange move occurs.

On multiple processors, the APTMC algorithm (referred to as `APTMC-W`) is implemented similarly. A number of $W = \Lambda = 8$ processors is used, where each worker $w = \lambda$ contains $K = 2$ chains, all targeting the same π_λ for $\lambda = 1, \ldots, \Lambda$. The multiple processor algorithm is run for $T = 10^7$ units of virtual time, with exchange moves alternating between occurring on all odd $(1, 2), (3, 4), (5, 6), (7, 8)$ and all even $(2, 3), (4, 5), (6, 7)$ pairs of workers every $\delta = 5$ units of virtual time. On each worker, the chain which was not working when calculations were interrupted is the one included in the exchange moves.

5.1.2 Verification of bias correction

To check that the single and multiple processor algorithms are successfully correcting for bias, they are also run *uncorrected*, i.e. not excluding the currently working chain. This means that exchange moves are also performed on samples distributed according to α instead of π, thus causing the algorithm to yield biased results. Since the bias is introduced by the exchange moves (when they are performed on α), we attempt to create a 'worst case scenario', i.e. maximise the amount of bias present when the single processor algorithm is uncorrected. The algorithm is further adjusted such that local moves are not performed on the cold chain and it is instead solely made up of samples resulting from exchange moves with the warmer chains. The multiprocessor `APTMC-W` algorithm is not run in a 'worst case scenario', so local moves on the cold chain of the multiprocessor algorithm are therefore allowed. This is meant to reveal how the bias caused by failing to correct when performing exchange moves across workers is still apparent, if less strongly.

Figure 3 shows kernel density estimates of the post burn-in cold chains resulting from runs of the `APTMC-1` and `APTMC-W` algorithms, uncorrected and corrected for bias. As expected, when the hold time does not depend on x, which corresponds to the case there $p = 0$, no bias is observed. On

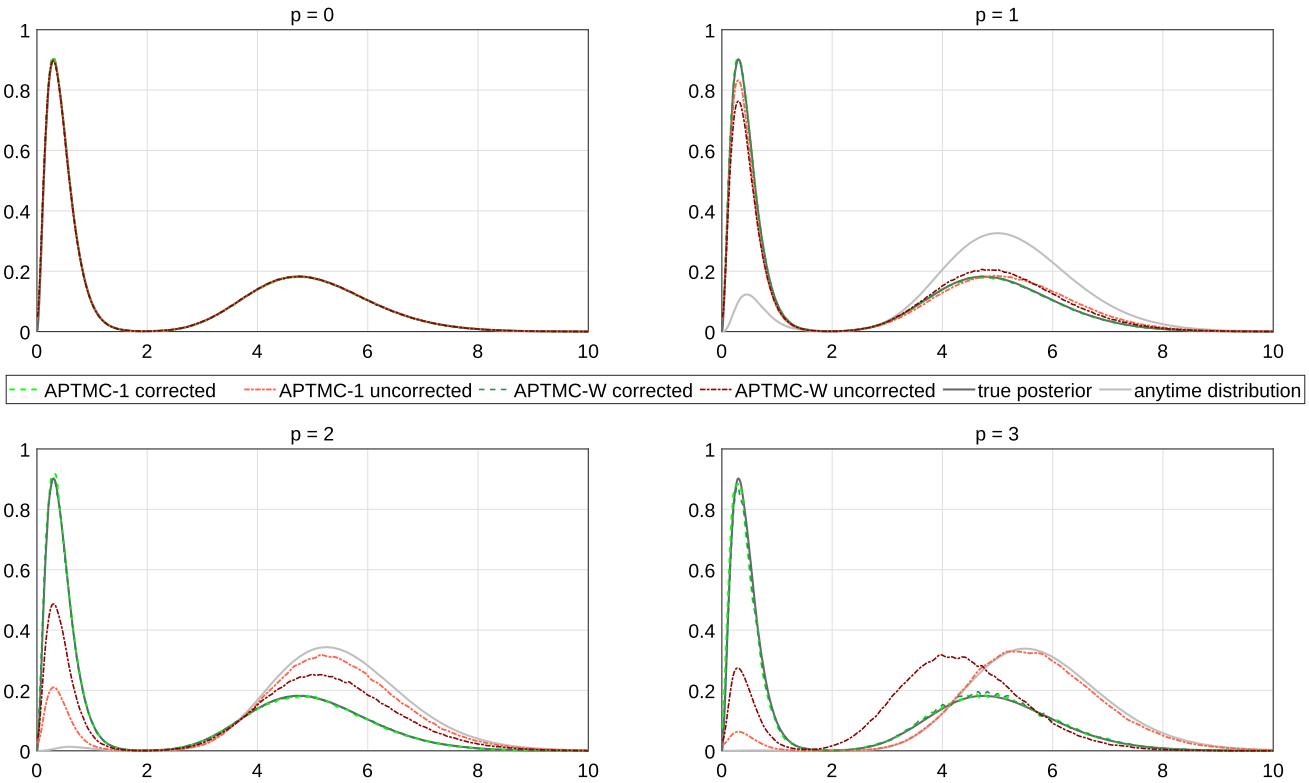

Fig. 3 Density estimates of the cold chain for bias corrected and uncorrected runs of the single (`APTMC-1`) and multi-processor (`APTMC-W`) algorithms on various hold time distributions $p \in \{0, 1, 2, 3\}$. In the single-processor case, the cold chain is made up entirely of updates resulting from exchange moves. The solid *dark grey* line represents the true posterior density π and the solid *light grey* line the anytime distribution α. The case $p = 0$ represents an instance in which, in a real-time situation, the local moves do not take a random time to complete, and therefore all densities are identical. The two *green* dashed lines represent bias corrected densities and the *red* dot-dashed represent uncorrected densities. For $p \geq 1$, the two corrected densities are identical to the posterior, indicating that the bias correction was successful. (Color figure online)

the other hand, the cold chains for the single-processor algorithm with computational complexity $p \in \{1, 2, 3\}$ have been corrupted by biased samples and converged to a shifted distribution which puts more weight the second Gamma mixture component, instead of an equal weight. Additionally, the bias becomes stronger as computational complexity p increases. A similar observation can be made for the cold chains from the multi-processor experiment – which display a milder bias due to local moves occurring on the cold chain. The green dashed densities indicate that when the algorithms are corrected, i.e. when the currently working chain is not included in exchange moves, it successfully eliminates the bias for all $p \in \{1, 2, 3\}$ to return the correct posterior π – despite even this being the 'worst case scenario' in the case of the `APTMC-1` algorithm. Note that the uncorrected density estimates do not exactly correspond to the anytime distributions. This has nothing to do with burn-in, but with the proportion of biased samples (from exchange moves) present in the chain.

5.1.3 Performance evaluation

Next we verify that introducing the parallel tempering element to the Anytime Monte Carlo algorithm improves performance. A standard `MCMC` algorithm is run for computational complexities $p \in \{0, 1, 2, 3\}$, applying the random walk Metropolis update described in Sect. 5.1.1. The single and multiple processor APTMC algorithms are run again for the same amount of virtual time, with exchange moves occurring every $\delta_{0:2} = 5$ units of virtual time for $p \leq 2$ and every $\delta_3 = 30$ units when $p = 3$. The single processor version is run on $\Lambda_s = 8$ chains, and the multi-processor on $W = 8$ workers, with $K = 2$ chains per worker, so $\Lambda_m = 16$ chains in total. This time, local moves are performed on the cold chain of the single processor `APTMC-1` algorithm.

To compare results, kernel density estimates of the posterior are obtained from the post burn-in cold chains for each algorithm using the `kde` function in MATLAB (2019), developed by Botev et al. (2010). It is also important to note that even though all algorithms run for the same (virtual) duration, the standard `MCMC` algorithm is performing local

Table 1 Integrated autocorrelation time (IAT) and effective sample size (ESS) for runs of the single, multi-processor Anytime parallel tempering and standard MCMC algorithms. The algorithms were run for 10^6 units of virtual time for computational complexity $p = 0$ and 10^7 units for $p \geq 1$, and the resulting ESSs were scaled down for consistency with $p = 0$. The ESS and IAT values for chains that have not converged to their posterior (their resulting kernel density estimates significantly under or overestimate modes in Fig. 4) have been omitted

p	Multi-processor		Single-processor		Standard	
	APTMC-W		APTMC-1		MCMC	
	IAT	ESS	IAT	ESS	IAT	ESS
0	53.925	12049	81.156	1202.2	1739.0	287.46
1	45.942	5888.3	95.104	708.74	2818.2	64.047
2	80.871	1168.4	132.79	448.92	–	–
3	131.91	116.51	–	–	–	–

moves on a single chain uninterrupted until the deadline, while the APTMC-1 algorithm has to update $\Lambda = 8$ chains in sequence, and each worker w of the APTMC-W algorithm has to update $K = 2$ chains in sequence before exchange moves occur. Therefore, by (virtual) time T the algorithms will not have returned samples of similar sizes. For a fair performance comparison, the sample autocorrelation function (acf) is estimated first of all. When available, the acf is averaged over multiple chains to reduce variance in its estimates. Other tools employed are

– *Integrated Autocorrelation Time* (IAT), the computational inefficiency of an MCMC sampler. Defined as

$$IAT_s = 1 + 2\sum_{\ell=1}^{\infty} \rho_s(\ell),$$

where $\rho_s(\ell)$ is the autocorrelation at the ℓ-th lag of chain s. It measures the average number of iterations required for an independent sample to be drawn, or in other words the number of correlated samples with same variance as one independent sample. Hence, a more efficient algorithm will have lower autocorrelation values and should yield a lower IAT value. Here, the IAT is estimated using a method initially suggested in Sokal (1997) and Goodman and Weare (2010), and implemented in the Python package emcee by Foreman-Mackey et al. (2013) (Section 3). Let

$$\hat{IAT}_s = 1 + 2\sum_{\ell=1}^{M} \hat{\rho}_s(\ell),$$

where M is a suitably chosen cutoff, such that noise at the higher lags is reduced. Here, the smallest M is chosen such that $M \geq C\hat{\rho}_s(M)$ where $C \approx 6$. More information on the choice of C is available in Sokal (1997).

– *Effective Sample Size* (ESS), the amount of information obtained from an MCMC sample. It is closely linked to the IAT by definition:

$$ESS_s = \frac{N_s}{IAT_s},$$

where N_s is the size of the current sample s. The ESS measures the number of independent samples obtained from MCMC output.

As per Foreman-Mackey et al. (2013), when multiple repeat runs of an experiment are performed (see Sect. 5.3), the IAT for a given algorithm is obtained by averaging the acf returned by this algorithm over the repeat runs, and the resulting ESSs of each run are summed to obtain a cumulative ESS for this algorithm.

The resulting ESS and IAT for different algorithms and computational complexities are computed and shown in Table 1. If an exchange move is accepted, the new state of the chain does not depend on the value of the previous state. This means that the autocorrelation in a chain containing a significant proportion of (accepted) samples originating from exchange moves will be lower. For low p, significantly more local moves occur before each deadline, as hold times are short, while for a higher p, the hold times are longer and hence fewer local moves are able to occur. Therefore, higher values of p will yield a higher proportion of samples from exchange moves, and thus a more notable increase in efficiency.

In Fig. 4 we observe, that the quality of the posterior estimates decreases as p increases. As a matter of fact, 10^7 units of virtual time tend to not be enough for the some of the posterior chains to completely converge. Indeed, while the standard MCMC algorithm performs reasonably well for $p = 0$, it becomes increasingly harder for it to fully converge for higher computational complexities. Similarly, the single processor APTMC-1 algorithm returns reasonably accurate posterior estimates for $p \leq 2$ but then visibly underestimates the first mode of the true posterior for $p = 3$. In general, the multi-processor APTMC-W algorithm returns results closest to the true cold posterior for all p.

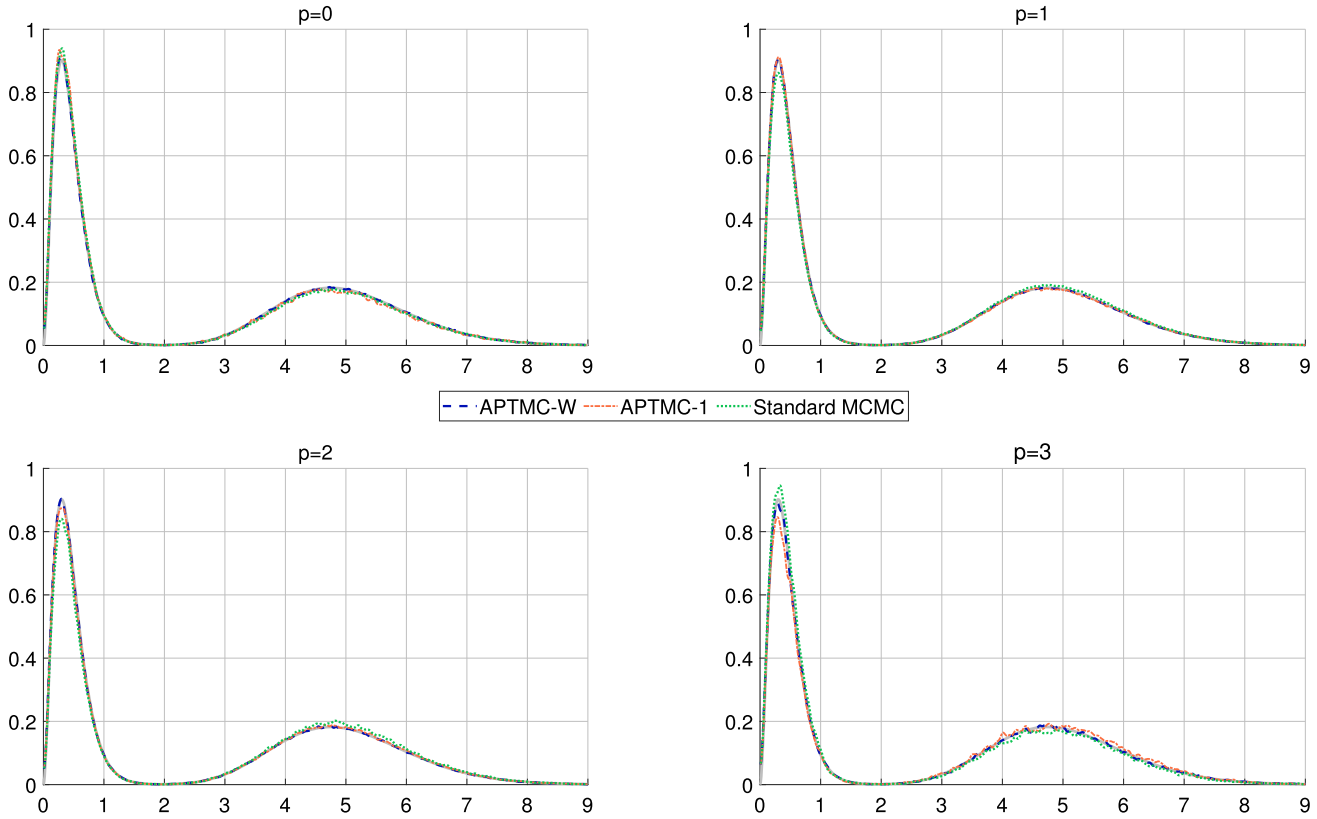

Fig. 4 Density estimates of the cold posterior for runs of the single (*orange*) and multiple (*blue*) processor APTMC algorithms (`APTMC-1` and `APTMC-W`, respectively) as well as the standard (*green*) `MCMC` algorithm. The *grey* line represents the true posterior density π. Each plot corresponds to a different hold time distribution $p \in \{0, 1, 2, 3\}$. While the multi processor density has successfully converged for all p − as evidenced by the perfect overlap between the grey and dark blue lines −, the other two algorithms tend to struggle more and more to estimate the first mode of the posterior as p increases. (Color figure online)

As for efficiency, Table 1 displays a much lower IAT and much higher ESS for both `APTMC` algorithms, indicating that they are much more efficient than the standard `MCMC` algorithm. This is further supported by the sample autocorrelation decaying much more quickly for `APTMC` algorithms than for the `MCMC` algorithm for all p in Fig. 5. The multi-processor `APTMC-W` algorithm also yields IAT values that are lower than those returned by the single processor `APTMC-1` algorithm for $p < 3$, and similarly yields ESSs that are higher for all p. The ESS and IAT values for chains that have not converged to their posterior (their resulting kernel density estimates significantly under or overestimate modes in Fig. 4) have been omitted from the table.

Next, we consider an application of the APTMC framework to ABC, a class of algorithms that are well-adapted to situations in which the likelihood is either intractable or computationally prohibitive. ABC features a real hold time at each MCMC iteration, making it an ideal candidate for adaptation to the Anytime parallel tempering framework.

5.2 ABC example: univariate Normal distribution

To validate the results of Sect. 4.2, consider another simple example, initially featured in Lee (2012), and adapted here within the APTMC framework. Let Y be a Gaussian random variable, i.e. $Y \sim \mathcal{N}(\theta, \sigma^2)$, where the standard deviation σ is known but the mean θ is not. The ABC likelihood here is

$$f^{\varepsilon}(y \mid \theta) = \Phi\left(\frac{y + \varepsilon - \theta}{\sigma}\right) - \Phi\left(\frac{y - \varepsilon - \theta}{\sigma}\right)$$

for $\varepsilon > 0$ where $\Phi(\cdot)$ is the cumulative distribution function (cdf) of a standard Gaussian. Using numerical integration tools in MATLAB, it is possible to obtain a good approximation of the true posterior for any ε for visualisation. Let $y = 3$ be an observation of Y and $\sigma^2 = 1$, and put the prior $p(\theta) = \mathcal{N}(0, 5)$ on θ. In this example, the exact posterior distribution for θ can straightforwardly be shown to be $\mathcal{N}\left(\frac{5}{2}, \frac{5}{6}\right)$.

When performing local moves (Algorithm 4), use a Gaussian random walk proposal with standard deviation $\xi = 0.5$.

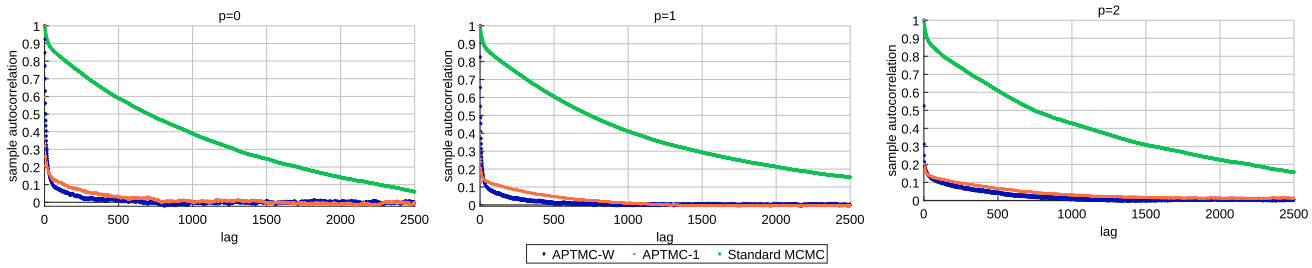

Fig. 5 Plots of the sample autocorrelation function up to lag 2500 of the post burn-in cold chain for runs of the single (*orange*) and multiple (*blue*) processor APTMC algorithms (`APTMC-1` and `APTMC-W`, respectively) as well as for the output of the standard Anytime Monte Carlo (`MCMC`) algorithm (*green*). Each plot corresponds to a different computational complexity $p \in \{0, 1, 2\}$. The two `APTMC` algorithms perform considerably better than standard `MCMC` for all p. The sample acf plot for $p = 3$ has been omitted due to both the `APTMC-1` and `MCMC` chains not having fully converged to their posterior. (Color figure online)

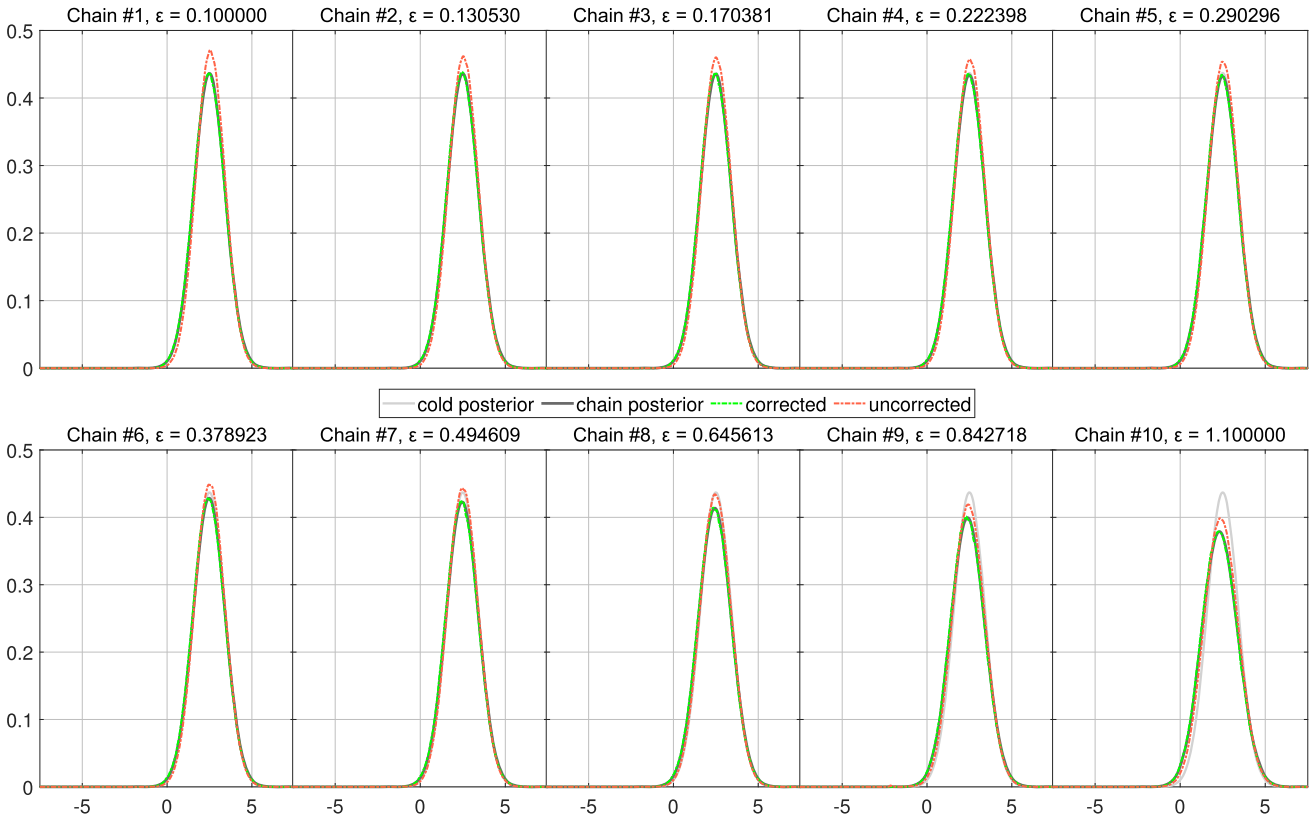

Fig. 6 Kernel density estimates of all chains for corrected and uncorrected runs of the single processor `ABC-APTMC` algorithm. In each subplot, the *light grey* line is fixed and represents the cold posterior for reference, the *dark grey* line represents each chain's target posterior (obtained by numerical integration), the dot-dashed *green* lines are kernel density estimates of the chain's posterior returned by the corrected algorithm and are indistinguishable from the dark grey line. The *orange* lines are kernel density estimates for the uncorrected algorithm, and do not agree with the dark grey line, as expected. (Color figure online)

The real-time Markov jump process is run using $\Lambda = 10$ chains. The algorithm is run on a single processor for one hour or $T = 3600$ seconds in real time after a 30 second burn-in, with exchange moves occurring every $\delta_T = 5 \times 10^{-4}$ seconds (or 0.5 milliseconds). The radii of the balls $\varepsilon^{1:\Lambda}$ are defined to vary between $\varepsilon^1 = 0.1$ and $\varepsilon^\Lambda = 1.1$.

We verify that bias correction must be applied for all chains to converge to the correct posterior. This is done by visually comparing density estimates of each of the post burn-in chains to the true corresponding posterior (obtained by numerical integration). When bias correction is not applied, the `ABC-APTMC` algorithm does not exclude the currently working chain j in its exchange moves. In this case, every chain converges to an erroneous distribution which overestimates the mode of its corresponding posterior, as is visible in Fig. 6. On the other hand, correcting the algorithm for

Table 2 Algorithm information and settings for stochastic Lotka-Volterra predator-prey model on a single processor

Label	Workers W	Chains Λ	Chains per worker K	Exchange moves (every)	Anytime
`ABC`	1	1	1	None	No
`ABC-PTMC-1`	1	6	6	6 local moves	No
`ABC-APTMC-1`	1	6	6	2.59 seconds	Yes

such bias ensures that every chain converges to the correct corresponding posterior.

Next, we compare the performance of the `ABC-APTMC` algorithm to that of a standard ABC (referred to as standard `ABC`) algorithm. For that, a more applied parameter estimation example is considered, for which the adoption of a likelihood-free approach is necessary.

5.3 Stochastic Lotka-Volterra model

In this section, we consider the stochastic Lotka-Volterra predator-prey model (Lotka (1926), Volterra (1927)), a model which has been explored in the past (Wilkinson (2011); Boys et al. (2008)), including in an ABC setting (Lee and Łatuszyński (2014); Fearnhead and Prangle (2012); Toni et al. (2009); Prangle et al. (2017)). Let $X_{1:2}(t)$ be a bivariate, integer-valued pure jump Markov process with initial values $X_{1:2}(0) = (50, 100)$, where $X_1(t)$ represents the number of prey and $X_2(t)$ the number of predators at time t. For small time interval Δt, we describe the predator-prey dynamics in the following way:

$$\mathbb{P}\{X_{1:2}(t+\Delta t) = z_{1:2} \mid X_{1:2}(t) = x_{1:2}\} = \begin{cases} \theta_1 x_1 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1+1, x_2), \\ \theta_2 x_1 x_2 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1-1, x_2+1), \\ \theta_3 x_2 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1, x_2-1), \\ o(\Delta t), & \text{otherwise.} \end{cases}$$

In this example, the only observations available are the number of prey, i.e. X_1 at 10 discrete time points. Following theory in Wilkinson (2011) (Chapter 6), the process can be simulated and discretised using the Gillespie (1977) algorithm, in which the inter-jump times follow an exponential distribution. The observations employed were simulated in Lee and Łatuszyński (2014) with true parameters $\theta = (1, 0.005, 0.6)$, giving $y = \{88, 165, 274, 268, 114, 46, 32, 36, 53, 92\}$ at times $\{1, \ldots, 10\}$.

This case study presents various challenges. The first challenge is that the posterior is intractable, and some of the components of the parameters $\theta := \theta_{1:3}$ (namely θ_2 and θ_3) exhibit strong correlations. Therefore, ABC must be employed, and the 'ball' considered takes the following form for $\varepsilon > 0$:

$$B_\varepsilon(y) = \{X_1(t) : |\log[X_1(i)] - \log[y(i)]| \leq \varepsilon, \forall i = 1, \ldots, 10\}. \quad (9)$$

So a set of simulated $X_1(t)$ is considered as 'hitting the ball' if all 10 simulated data points are at most e^ε times (and at least $e^{-\varepsilon}$ times) the magnitude of the corresponding observation in y.

In Lee and Łatuszyński (2014) (Algorithm 3), the 1-hit MCMC kernel (referred to here as standard `ABC`) is shown to return the most reliable results by comparison with other MCMC kernels, which are not considered here. The second challenge is that while this algorithm can be reasonably fast, it is highly inefficient as it has a very low acceptance rate, and thus the autocorrelation between samples for low lags is very high.

We have established that the race step in Algorithm 4 takes a random time to complete. In addition to that, its hold time distribution for the Lotka-Volterra model is a mixture between quick and lengthy completion times, as the simulation steps within the 1-hit kernel race are capable of taking a considerable amount of time despite often being almost instant. Indeed, when simulating observations using the discretised Gillespie algorithm, if the number of predators is low, prey numbers will flourish and the simulation will take longer. Hence, the third challenge in this particular model is that the race can sometimes get stuck for extended periods of time if the number of prey to simulate is especially high. Therefore, we aim to first of all improve performances by introducing exchange moves on a single processor (`ABC-PTMC`). Then – and most importantly – we further improve the algorithm by implementing it within the Anytime framework (`ABC-APTMC`), a method which works especially well on multiple processors.

5.3.1 One processor

The first part of this case study is run on a single processor and serves to demonstrate the gain in efficiency introduced by the exchange moves described in Algorithm 5. Define the prior on $\theta \in [0, \infty)^3$ for the single processor experiment to be $p(\theta) = \exp\{-\theta_1 - \theta_2 - \theta_3\}$. The proposal distribution is a truncated normal, i.e. $\theta' \mid \theta \sim \mathcal{TN}(\theta, \Sigma)$, $\theta' \in (0, 10)$

with mean θ and covariance $\Sigma = \text{diag}\,(0.25, 0.0025, 0.25)$. The truncated normal is used in order to ensure that all proposals remain non-negative. For reference, 2364 independent samples from the posterior are obtained via ABC rejection sampling with $\varepsilon = 1$ and the density estimates in Figure 6 of Lee and Łatuszyński (2014) are reproduced. To obtain these posterior samples, 10^7 independent samples from the prior were required, yielding the very low 0.024% acceptance rate. This method of sampling from the posterior is therefore extremely inefficient, and the decision to resort to MCMC kernels in order to improve efficiency is justified.

On a single processor, the three algorithms considered are the vanilla 1-hit MCMC kernel (standard `ABC`) defined in Algorithm 4, the single processor version of the algorithm with added exchange moves (`ABC-PTMC-1`) and the same but within the Anytime framework (`ABC-APTMC-1`). They are run nine times for 100800 seconds (28 hours) − after 3600 seconds (1 hour) of burn-in − and their main settings are summarised in Table 2.

Given the aim is to compare the performance of these algorithms, it is important to note that the parallel tempering algorithms, having to deal with updating multiple chains sequentially, are likely to return cold chains with fewer samples. The algorithms must therefore be properly set up such that the gain in efficiency introduced by exchange moves is not overshadowed by the greater number of chains and computational cost of having to update them all. In this experiment, the parallel tempering algorithms are run on $\Lambda = 6$ chains, each targeting posteriors associated with balls of radii $\varepsilon^{1:6} = \{1, 1.1447, 1.3104, 1.5, 11, 15\}$ and the proposal distribution has covariance $\Sigma^{1:6}$ where $\Sigma^{\lambda} = \text{diag}\left(\sigma^{\lambda}, \sigma^{\lambda}10^{-2}, \sigma^{\lambda}\right)$ and $\sigma^{1:6} = \{0.008, 0.025, 0.05, 0.09, 0.25, 0.5\}$. Exchange moves are performed as described in Algorithm 5 and alternate between odd $(1, 2), (3, 4), (5, 6)$ (excluding $(5, 6)$ in the Anytime version) and even $(2, 3), (4, 5)$ pairs of eligible chains. As per Sect. 3.4, exchange moves for the `ABC-PTMC-1` algorithm are performed every $\Lambda = 6$ local moves, and the real-time deadline δ for exchange moves in the `ABC-APTMC-1` algorithm is determined by repeatedly measuring the times taken by the `ABC-PTMC-1` algorithm to perform these 6 moves and setting δ to be the median over all measured times.

5.3.2 Multiple processors

Next, we demonstrate the gain in efficiency introduced by running the parallel tempering algorithm within the Anytime framework on multiple processors. The algorithms considered are the multi-processor `ABC-PTMC-W` and `ABC-APTMC-W` algorithms and their single processor counterparts `ABC-PTMC-1` and `ABC-APTMC-1` . This time, we define a uniform prior between 0 and 3. The proposal distribution is still a truncated normal, but with tighter limits (corresponding to the prior) i.e. $\theta' \,|\, \theta \sim T\mathcal{N}(\theta, \Sigma)$, $\theta' \in (0, 3)$.

The two algorithms are run on $\Lambda = 20$ chains, each targeting posteriors associated with balls of radii ranging from $\varepsilon^1 = 1$ to $\varepsilon^{20} = 11$ and proposal distribution covariances $\Sigma^{1:20}$ where $\Sigma^{\lambda} = \text{diag}\left(\sigma^{\lambda}, \sigma^{\lambda}10^{-2}, \sigma^{\lambda}\right)$ for chain $\lambda = 1, \ldots, 20$ and where values range from $\sigma^1 = 0.008$ to $\sigma^{20} = 0.5$ (see Table 6). The algorithms are run four times for 864000 seconds (24 hours) and their main settings are summarised in Table 3. Given the non-negligible 1.1 second communication overhead, this experiment is run according to the third scenario from Sect. 3.3, i.e. dividing exchange moves into within- and between-worker exchange moves. As described in Sect. 3.4, a full set of parallel moves here consists of $K = 5$ local moves, followed by within-worker exchange moves between a pair of adjacent chains selected at random, followed by 5 more local moves. The between-worker exchange moves are performed after a full set of parallel moves on the master by selecting a pair of adjacent workers at random and exchanging between the warmest eligible chain from the first worker and coldest from the second so that they are adjacent.

5.3.3 Performance evaluation

All algorithms returned density estimates that were close to those obtained via rejection sampling. In order to compare the performance of the algorithms, they are set to run for the same real time period. The IAT and cumulative ESS over all repeat runs are computed for all algorithms. The ESS is particularly important here, as it gives us how many effective samples the different algorithms can return within a fixed time frame. For example, a very fast algorithm could still return a higher ESS even if it has a much higher IAT. Additionally, to illustrate how the Anytime version of the parallel tempering algorithms works compared to standard `ABC-PTMC`, the real times all algorithms take to perform local and exchange moves are measured and their timelines plotted in Fig. 8.

One processor

Both the `ABC-PTMC-1` and `ABC-APTMC-1` algorithm display an improvement in performances: they return IATs that are respectively 3.2 and 1.6 times lower on average than those of the standard `ABC` algorithm in Table 4, and display a steeper decay in sample autocorrelation in Fig. 7. In the 28 hours (post burn-in) during which the algorithms ran, both parallel algorithms also yielded an increased ESS. The effect of the Anytime framework on the behaviour of the parallel tempering algorithm is demonstrated in Fig. 8. The timeline of local moves for the `ABC-PTMC-1` algorithm illustrates the fact that local moves take a random amount of time to complete. In the Anytime version of the algorithm, this is mitigated since a deadline was implemented. As a result, the

Table 3 Algorithm information and settings for stochastic Lotka-Volterra predator-prey model on multiple processors

Label	Workers W	Chains Λ	Chains per worker K	Communication overhead	Exchange moves (every)	Anytime
ABC-PTMC-1	1	20	20	–	20 local moves	No
ABC-APTMC-1	1	20	20	–	11 seconds	Yes
ABC-PTMC-W	4	20	5	1.1 seconds	5 local moves	No
ABC-APTMC-W	4	20	5	1.1 seconds	5 local moves (*Within* workers) 15.3 seconds (*between* workers)	Yes

bottom plot in Fig. 8 displays more consistent local move times.

Note that in Table 4, while the improvement in IAT is significant, the increase in ESS after 28 hours is not particularly huge. This is due to the previously mentioned erratic behaviour of the hold time distribution for this example. Other examples explored such as the moving average example in Marin et al. (2012) (not reported here) yielded a much more significant increase in ESS after introducing exchange moves. We also note that in this example, the ABC-PTMC-1 algorithm returned a lower IAT than its Anytime counterpart but Anytime had a larger ESS. There are two potential reasons to account for the IAT. The first is the many swaps which are cycling the same samples among the held chains of Anytime. The second, as mentioned in Sect. 3.4, is that the Anytime algorithms cannot always exchange the samples of adjacent chains, because they must exclude the chain that is currently computing, and this causes a slightly higher rejection rate compared to the standard version (in the multi-processor example with more chains, this is mitigated). However, the many more swap moves of Anytime does then result in more returned samples, which leads to a higher ESS. The single processor experiment was mainly designed to demonstrate the performance improvements brought by adding exchange moves to the 1-hit MCMC kernel (referred to as standard ABC) and to show that Anytime does match the performance of parallel tempering on a single processor. Since a single processor is never idling, the strength of the Anytime framework is best illustrated in a multi-processor setting.

5.3.4 Multiple processors

In the multi-processor case study, both the ABC-PTMC-1 and ABC-PTMC-W were set so that on each worker, an exchange move occurred after all chains had been updated locally once, as described in Table 3. The total number of samples returned by the ABC-PTMC-W algorithm is higher for all chains (see Table 6). However, the ABC-PTMC-W algorithm is just as affected by the distribution of the hold times being a mixture of quick and lengthy completion times as its single processor counterpart, and is just as prone to getting stuck in a race for an extended period. During this time, all processors sit idle while waiting for the race to complete, as illustrated in Fig. 9. Therefore, the ABC-PTMC-W algorithm struggles to properly boost the total sample size output by the cold chain, and the ESS is not markedly higher on average in Table 5. On the other hand, thanks to the real time deadlines implemented, the ABC-APTMC-W algorithm is able to more than double the total size of the samples output (see Table 6), and the ESSs for the cold chain returned in Table 5 are on average 3.41 times higher than those of the single processor version.

As for the main comparison – namely Anytime vs standard ABC with exchange moves – both single and multi-processor Anytime algorithms return an ESS larger than their respective standard versions in Table 5. While the improvement on a single processor is modest, the ESS has more than quadrupled on multiple processors. Figures 9 and 10 illustrate well the advantage of implementing a real-time deadline to local moves. At most local moves, the issue in which all workers sit idle waiting for the slowest to finish arises for the ABC-PTMC-W algorithm. On the other hand, Fig. 10 shows that the Anytime version of the algorithm is making better use of the allocated computational resources. The Anytime framework ensures that none of the workers need to wait for the slowest among them to finish, allowing for more exploration of the sample space in the faster workers. Additionally, the real time deadline ensures that even if chain k on worker w remains stuck in a race for an extended period of time, the other workers are still updating. Therefore, while the remaining four chains on worker w wait for chain k to complete its race, they also continue to be updated at regular intervals thanks to the exchange moves with other workers. The addition of ABC exchange moves in his case study proved fruitful, as the ESS for the parameters of the Lotka-Volterra model was increased.

6 Conclusion

In an effort to increase the efficiency of MCMC algorithms, in particular for use on multiple processors, and for situations in which compute times of the algorithms depend on their

Table 4 Integrated autocorrelation time (IAT) and cumulative effective sample size (ESS) over nine 28-hour runs of the standard ABC, ABC-PTMC-1 and ABC-APTMC-1 algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model. Improvements in performance are modest in this example. (Color figure online)

	Standard ABC		ABC-PTMC-1		ABC-APTMC-1	
	IAT	ESS	IAT	ESS	IAT	ESS
θ_1	69.476	7018.1	22.404	7618.6	44.071	7963.8
θ_2	122.73	3973	35.381	4824.2	69.803	5028
θ_3	150.74	3234.6	50.035	3411.3	98.929	3547.7

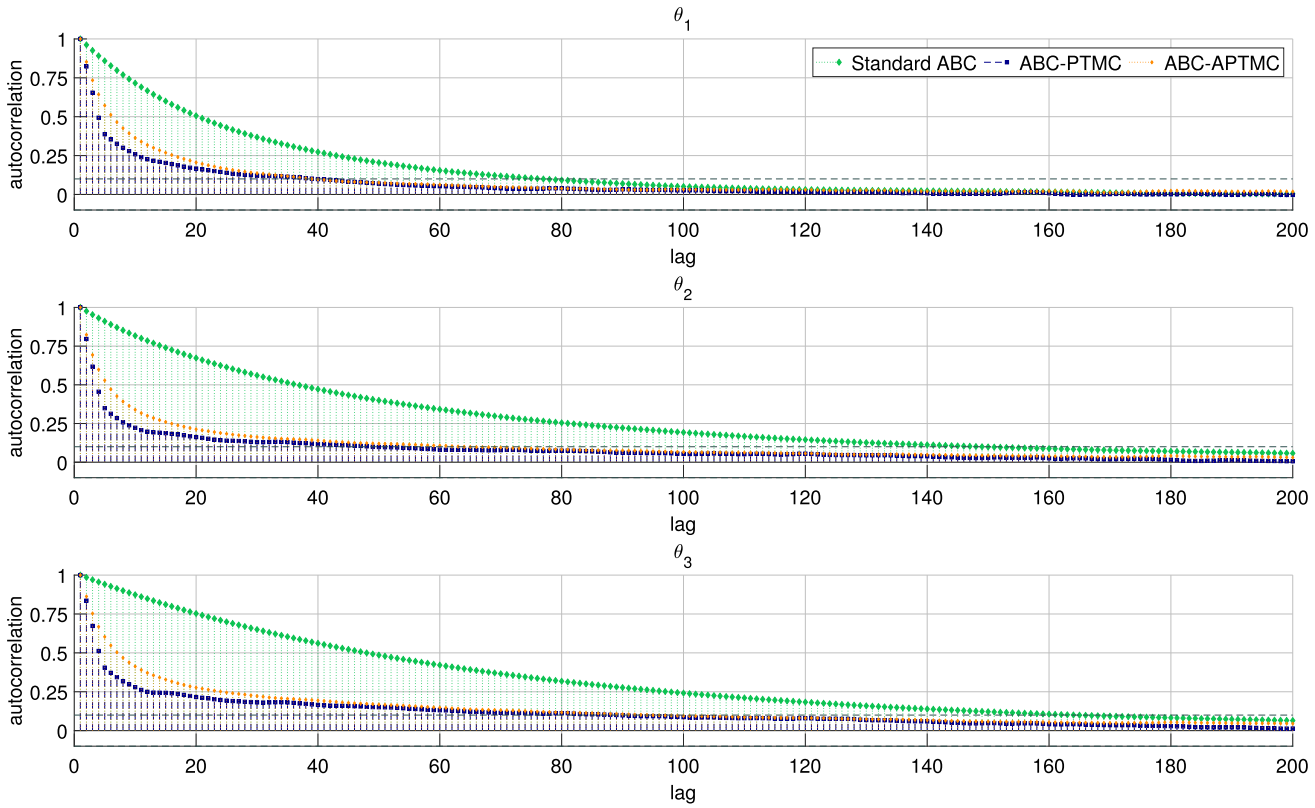

Fig. 7 Plots of the sample autocorrelation function up to lag 200 of the cold chain for runs of the standard ABC (*green*), ABC-PTMC-1 (*blue*) and ABC-APTMC-1 (*orange*) algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model. The inclusion of exchange moves boosts efficiency and leads to a steeper decay in the parallel tempering algorithms. (Color figure online)

current states, the APTMC algorithm was developed. The algorithm combines the enhanced exploration of the state space, provided by the between-chain exchange moves in parallel tempering, with control over the real-time budget and robustness to interruptions available within the Anytime Monte Carlo framework. Then, an application of APTMC to problems where the likelihood is unavailable and an ABC MCMC kernel, in particular the 1-hit MCMC kernel, must be employed instead was considered.

Initially, the construction of the Anytime Monte Carlo algorithm with the inclusion of exchange moves on a single and multiple processors was verified on a Gamma mixture example. The performance improvements they brought were then demonstrated by comparing the algorithm to a standard MCMC algorithm. Subsequently, the exchange moves were adapted for pairing with the 1-hit MCMC kernel, a simulation-based algorithm within ABC framework, which provides an attractive, likelihood-free approach to MCMC. The construction of the adapted ABC algorithm was verified using a univariate normal example. Then, the increased efficiency of the inclusion of exchange moves was demonstrated in comparison to that of a standard ABC algorithm on a parameter estimation problem. The problem involved the parameters of a stochastic Lotka-Volterra predator-prey model based on partial and discrete data, and the likelihood of this model is intractable. On a single processor, it was shown

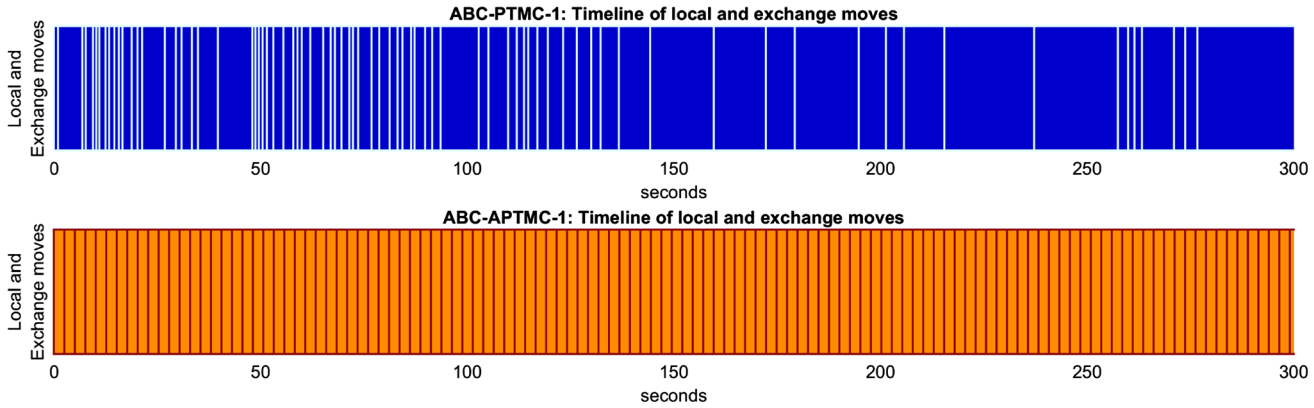

Fig. 8 Timeline of local and exchange moves for the `ABC-PTMC-1` and `ABC-APTMC-1` algorithms for the first 300 seconds. The exchange moves are represented by the *white* and *red* lines and the local moves by the *dark blue* and *orange* coloured blocks. Local moves take a random amount of time to complete, as illustrated by the times consumed by local moves for the `ABC-PTMC-1` algorithm. The Anytime (`ABC-APTMC-1`) version effectively implements a hard deadline for the exchange moves (without introducing a bias), as can be seen by the regularity of local move times in the bottom figure. (Color figure online)

Table 5 Integrated autocorrelation time (IAT) and cumulative effective sample size (ESS) over four 24-hour runs of the `ABC-PTMC-1`, `ABC-APTMC-1`, `ABC-PTMC-W` and `ABC-APTMC-W` algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model

	One processor				**Multiple processors**			
	`ABC-PTMC-1`		`ABC-APTMC-1`		`ABC-PTMC-W`		`ABC-APTMC-W`	
	IAT	ESS	IAT	ESS	IAT	ESS	IAT	ESS
θ_1	39.535	269.89	72.475	362.62	48.621	266.7	39.898	1452.5
θ_2	72.908	146.35	88.446	297.14	67.395	192.4	72.79	796.14
θ_3	82.464	129.39	138.56	189.68	87.635	147.97	101.57	570.57

that introducing exchange moves provides an improvement in performance and an increase in the effective sample size compared to that of the standard, single chain ABC algorithm. The Anytime results for a single processor matches the efficiency of standard parallel tempering, which is to be expected since the single processor is never idling in both the Anytime and non-Anytime versions. The ESS gains of Anytime become significant in a multi-processor setting since one slow processor will lead to all the others idling in standard parallel tempering.

One major class of applications with local moves that have state-dependent real completion times and could benefit from the APTMC framework are transdimensional applications, such as RJ-MCMC (Green (1995)), which has a parallel tempering implementation in Jasra et al. (2007). The performance of parallel tempering algorithms with temperature-dependent completion times, as addressed in Earl and Deem (2004), can also be improved by the Anytime framework. Examples of such algorithms occur in Hritz and Oostenbrink (2007); Karimi et al. (2011); Wang and Jordan (2003). From a purely computing infrastructure-related perspective, exogenous factors such as processor hardware, competing jobs, memory bandwidth, network traffic or I/O load also affect the completion time of local moves even if they are not state-dependent within the algorithm itself. This is the case in Rodinger et al. (2006). In a more general setting, any population-based MCMC algorithm such as parallel tempering, SMC samplers (Del Moral et al. (2006)), or parallelised generalised elliptical slice sampling (Nishihara et al. (2014)), which combines a parallel updates step (e.g. local moves) and an inter-processor communication step (e.g. exchange moves, resampling) can benefit from the APTMC framework in future implementations.

As a final comment, we note the potential relevance of the work of Dupuis et al. (2012) in studying efficiency as a function of exchange frequency. As exchange steps of Anytime parallel tempering become more frequent, i.e. many occur between the stalled chains before the local move completes, it would be interesting to explore if our Anytime parallel tempering algorithm could be understood in the framework of the *infinite* swapping limit version of parallel tempering which has been shown in Dupuis et al. (2012) to dominate in numerical examples and in a specific theoretical context. However, their analysis ignores the cost of performing exchanges,

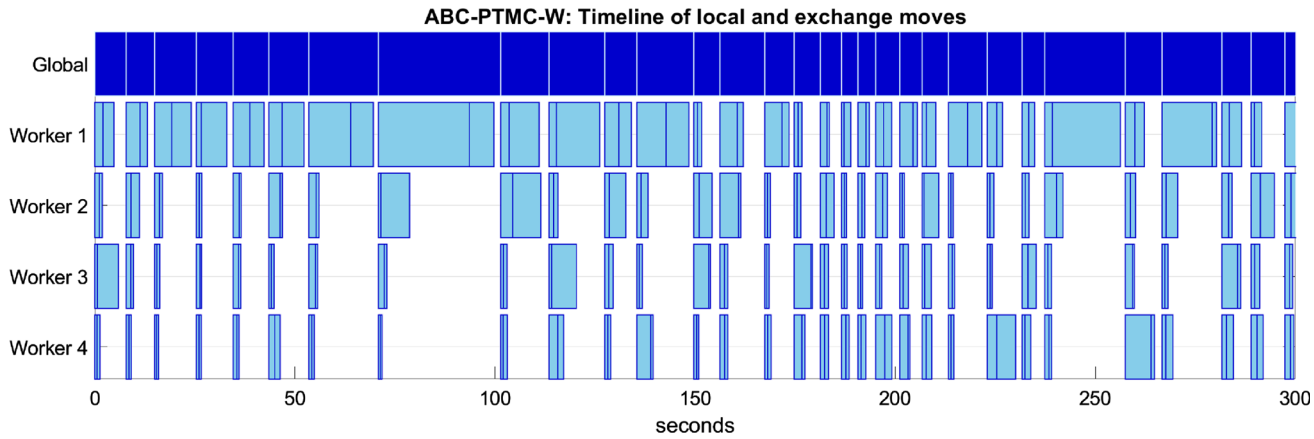

Fig. 9 Timeline of local and exchange moves for the `ABC-PTMC-W` algorithm for the first 300 seconds. Within and between worker exchange moves are represented by the *white lines* on the Global timeline and *blue lines* on the various Worker timelines, respectively. Local moves on each worker are represented by the *light blue* coloured blocks and the *dark blue* coloured blocks correspond to the global time all workers spend running in parallel, including communication overhead. Significant idle time is apparent on all workers as they always have to wait for the slowest among them to complete its set of local moves. (Color figure online)

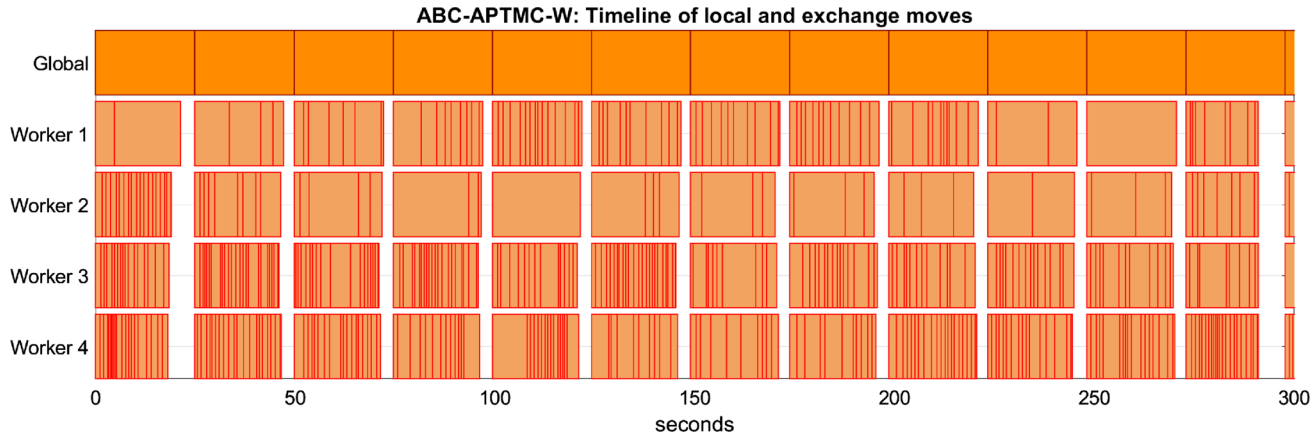

Fig. 10 Timeline of local and exchange moves for the `ABC-APTMC-W` algorithm for the first 300 seconds. Within and between worker exchange moves are represented by the *red lines*. Local moves on each worker are represented by the various *orange* coloured blocks, with the brighter blocks corresponding to the global time all workers spend running in parallel, including communication overhead. The significant idle time from Fig. 9 has been greatly reduced thanks to the deadlines implemented

which is non-negligible when communicating across processors, and thus cannot be plainly advocated without more consideration.

Acknowledgements This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC). We thank the Reviewers for very helpful comments that helped us improve the paper.

A Proofs

A.1 Proof of Proposition 2

The continuous time chain of Proposition 2 is described in steps 1 to 6 (excluding the exchange steps) of Algorithm 1. The Markov transition kernel of this chain is

Table 6 Average sample sizes per chain returned over four 24-hour runs of the `ABC-PTMC-1`, `ABC-APTMC-1`, `ABC-PTMC-W`, `ABC-APTMC-W` algorithms to estimate the posterior distributions of the parameters θ of a stochastic Lotka-Volterra model on multiple processors in Sect. 5.3.3. The ball radius ε^k and proposal distribution covariance $\mathrm{diag}(\sigma^k, \sigma^k 10^{-2}, \sigma^k)$ associated with each chain k are displayed for information

Chain k	ε^k	σ^k	`ABC-PTMC-1`	`ABC-PTMC-W`	`ABC-APTMC-1`	`ABC-APTMC-W`
1	1	0.008	2667.5	3241.8	6570.3	14488
2	1.046	0.009	2790	3564.8	6934.8	16902
3	1.094	0.011	2796.8	3567.5	6941	16837
4	1.145	0.012	2797.3	3604.5	6924.8	17264
5	1.197	0.014	2793.8	3719.8	6931.3	15710
6	1.253	0.016	2786.3	3748.8	6951.5	17157
7	1.31	0.019	2784.8	3629.3	6961.3	18947
8	1.371	0.022	2795.5	3615	6941.5	18551
9	1.434	0.025	2805.5	3608.5	6950.8	18759
10	1.5	0.029	2803.5	3711.5	6962.8	17276
11	1.661	0.034	2798.5	3799.8	6962.3	46350
12	1.84	0.039	2803.3	3693.3	6983	53716
13	2.038	0.045	2814.8	3656.5	6995.3	53289
14	2.257	0.052	2799	3658.3	7008.8	53458
15	2.5	0.06	2783.5	3796.3	7029.8	46597
16	3.362	0.092	2787.5	4054.5	7038.5	68953
17	4.522	0.14	2783.8	3936.5	7009.5	79231
18	6.082	0.214	2781	3912.5	6982.5	78917
19	8.179	0.327	2780.8	3919.5	7002.8	79038
20	11	0.5	2665.8	3598.5	6604.3	67725

$$
\begin{aligned}
&(P_t f)(x^{1:\Lambda}, l, j)\\
&= \mathbb{E}\left\{ f(X^{1:\Lambda}(t), L(t), J(t)) \middle| \left(X^{1:\Lambda}, L, J\right)(0) = \left(x^{1:\Lambda}, l, j\right)\right\}\\
&= \mathbb{E}\left\{ f(X^{1:\Lambda}(t), L(t), J(t)) \mathbb{I}_{\{L(t)\geq t\}} | x^{1:\Lambda}, l, j\right\}\\
&+ \mathbb{E}\left\{ f(X^{1:\Lambda}(t), L(t), J(t)) \mathbb{I}_{\{L(t)< t\}} | x^{1:\Lambda}, l, j\right\},
\end{aligned}
$$

where in the second line, the conditioning on the state at time 0 has been abbreviated, and the two events $\{L(t) \geq t\}$ and $\{L(t) < t\}$ have been introduced to simplify the calculation.

The event $\{L(t) \geq t\}$ implies that chain j hasn't completed its local move by time t. It follows then that

$$
\begin{aligned}
&\mathbb{E}\left\{ f(X^{1:\Lambda}(t), L(t), J(t)) \mathbb{I}_{\{L(t)\geq t\}} | x^{1:\Lambda}, l, j\right\}\\
&= f(x^{1:\Lambda}, l+t, j) \frac{\bar{F}_j(l+t|x^j)}{\bar{F}_j(l|x^j)},
\end{aligned}
$$

where $\bar{F}_j(l|x^j) = 1 - F_j(l|x^j)$ and $F_j(l|x^j)$ is the cdf of the hold time density of $\tau_j(h|x^j)\mathrm{d}h$ for chain j. Note that the conditioning on $(x^{1:\Lambda}, l, j)$ gives rise to the term $\bar{F}_j(l|x^j)$ in the denominator, and thus the ratio is the probability $\mathbb{P}\left(L(t) \geq t | x^{1:\Lambda}, l, j\right)$.

To simplify the calculation for the event $\{L(t) < t\}$, assume $t \leq \epsilon$ where ϵ is the assumed (in Sect. 2) minimum hold time. That is, the hold time (variable $L(t)$ here) exceeds $\epsilon > 0$ with probability 1. Thus, the event $\{L(t) < t\}$ for $t \leq \epsilon$ corresponds to a single possible scenario where chain j completes its local move at some time s in the time interval $(0, t]$, and thus holds for a total time of $l + s$. Chain $j + 1$ is next to be worked on, and is still being worked on at time t, thus $J(t) = j + 1$ and $L(t) = t - s$. Applying this reasoning, we have

$$
\begin{aligned}
&\mathbb{E}\left\{ f(X^{1:\Lambda}(t), L(t), J(t)) \mathbb{I}_{\{L(t)< t\}} | x^{1:\Lambda}, l, j\right\}\\
&= \int_0^t \left(\int (P_{t-s} f)(x^{1:j-1}, y, x^{j+1:\Lambda}, 0, j+1)\right.\\
&\left.\kappa_j(y|x^j, l+s)\mathrm{d}y\right) \times \frac{\tau_j(l+s|x^j)}{\bar{F}_j(l|x^j)}\mathrm{d}s,
\end{aligned}
$$

where the inner integral averages over the new state for chain $x^j \to y$ when the hold time is $l + s$, while other states $x^{1:j-1}$ and $x^{j+1:\Lambda}$ are unchanged. The outer integral averages over the hold time distribution conditioned on $L(0) = l$. The usual semigroup property (see Del Moral and Penev (2017) for general background) for a Markov process $(P_t f)(x^{1:\Lambda}, l, j) = (P_s(P_{t-s} f))(x^{1:\Lambda}, l, j)$, is also being employed.

A final simplification is applied to the integrand

$$
\begin{aligned}
&(P_{t-s} f)(x^{1:j-1}, y, x^{j+1:\Lambda}, 0, j+1)\\
&= f(x^{1:j-1}, y, x^{j+1:\Lambda}, t-s, j+1)
\end{aligned}
$$

since $t - s \leq \epsilon$ and thus the hold time of chain j advances to $t - s$ from 0.

Using the specific form of $A(\mathrm{d}x^{1:\Lambda}, \mathrm{d}l, j)$ given in Proposition 2 and the integrand $(P_t f)(x^{1:\Lambda}, l, j)$ developed above, gives the desired result, namely for any $t \leq \epsilon$, we have

$$\sum_{j=1}^{\Lambda} \int (P_t f)(x^{1:\Lambda}, l, j) A(\mathrm{d}x^{1:\Lambda}, \mathrm{d}l, j) = \sum_{j=1}^{\Lambda} \int f(x^{1:\Lambda}, l, j) A(\mathrm{d}x^{1:\Lambda}, \mathrm{d}l, j).$$

The results thus generalises to any $t > \epsilon$ by the semigroup property: $(P_{t+h} f) = P_h (P_t f)$.

A.2 Anytime distribution of the Gamma mixture cold chain

To obtain the anytime distribution in the Gamma mixture example in Sect. 5.1, compute the three components of the expression in Equation (4):

1. The density of X

$$\pi(\mathrm{d}x) = \frac{x^{k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}} e^{-\frac{x}{\theta_1}} + \frac{x^{k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}} e^{-\frac{x}{\theta_2}} \,\mathrm{d}x,$$

   where $\Gamma(\cdot)$ is the gamma function.
2. The expectation of $H \mid x$ given by

$$\mathbb{E}[H \mid x] = \psi x^p + (1-\psi) x^p = x^p.$$

   The ψ factors cancel out, meaning that the anytime distribution is independent of ψ and therefore its value can be chosen to be 1 for convenience.
3. To compute $\mathbb{E}[H]$, use a property of conditional expectation and the honesty conditions of the Gamma$(k_1 + p, \theta_1)$ and Gamma$(k_2 + p, \theta_2)$ distributions:

$$\begin{aligned}\mathbb{E}[H] &= \mathbb{E}[\mathbb{E}(H \mid x)] = \mathbb{E}\left[x^p\right] \\ &= \int \frac{x^{p+k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}} e^{-\frac{x}{\theta_1}} \,\mathrm{d}x + \int \frac{x^{p+k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}} e^{-\frac{x}{\theta_2}} \,\mathrm{d}x \\ &= \frac{\Gamma(k_2)\Gamma(p+k_1)\theta_1^p + \Gamma(k_1)\Gamma(p+k_2)\theta_2^p}{2\Gamma(k_1)\Gamma(k_2)} \\ &= \frac{C}{2\Gamma(k_1)\Gamma(k_2)},\end{aligned}$$

   letting $C = \Gamma(k_2)\Gamma(p+k_1)\theta_1^p + \Gamma(k_1)\Gamma(p+k_2)\theta_2^p$.

Combining the three components,

$$\begin{aligned}&\alpha(\mathrm{d}x) \\ &= \frac{2\Gamma(k_1)\Gamma(k_2)}{C}\left(\frac{x^{p+k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}} e^{-\frac{x}{\theta_1}} + \frac{x^{p+k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}} e^{-\frac{x}{\theta_2}}\right)\mathrm{d}x \\ &= \underbrace{\frac{\Gamma(k_2)\Gamma(p+k_1)\theta_1^{p+k_1}}{C\theta_1^{k_1}}}_{\varphi(p,k_{1:2},\theta_{1:2})} \underbrace{\frac{x^{p+k_1-1}}{\Gamma(p+k_1)\theta_1^{p+k_1}} e^{-\frac{x}{\theta_1}}}_{\text{Gamma}(p+k_1,\theta_1)} \\ &+ \underbrace{\frac{\Gamma(k_1)\Gamma(p+k_2)\theta_2^{p+k_2}}{C\theta_2^{k_2}}}_{\varphi'(p,k_{1:2},\theta_{1:2})} \underbrace{\frac{x^{p+k_2-1}}{\Gamma(p+k_2)\theta_2^{p+k_2}} e^{-\frac{x}{\theta_2}}}_{\text{Gamma}(p+k_2,\theta_2)} \,\mathrm{d}x.\end{aligned}$$

And now substituting back the expression C in φ:

$$\varphi(p, k_{1:2}, \theta_{1:2}) = \left(1 + \frac{\Gamma(k_1)\Gamma(p+k_2)\theta_2^p}{\Gamma(k_2)\Gamma(p+k_1)\theta_1^p}\right)^{-1}.$$

Similarly, we can obtain $\varphi'(p, k_{1:2}, \theta_{1:2}) = 1 - \varphi(p, k_{1:2}, \theta_{1:2})$. Therefore, the anytime distribution $\alpha(\mathrm{d}x)$ is the following mixture of two Gamma distributions:

$$\begin{aligned}\alpha(\mathrm{d}x) = \varphi(p, k_{1:2}, \theta_{1:2})\,&\text{Gamma}(k_1 + p, \theta_1) \\ &+ [1 - \varphi(p, k_{1:2}, \theta_{1:2})]\,\text{Gamma}(k_2 + p, \theta_2).\end{aligned}$$

.

References

Atchadé, Y.F., Roberts, G.O., Rosenthal, J.S.: Towards optimal scaling of Metropolis-coupled Markov chain Monte Carlo. Stat. Comput. **21**(4), 555–568 (2011)

Baragatti, M., Grimaud, A., Pommeret, D.: Likelihood-free parallel tempering. Stat. Comput. **23**(4), 535–549 (2013)

Beskos, A., Roberts, G., Stuart, A., et al.: Optimal scalings for local Metropolis-Hastings chains on nonproduct targets in high dimensions. Ann. Appl. Probab. **19**(3), 863–898 (2009)

Botev, Z.I., Grotowski, J.F., Kroese, D.P., et al.: Kernel density estimation via diffusion. Ann. Stat. **38**(5), 2916–2957 (2010)

Boys, R.J., Wilkinson, D.J., Kirkwood, T.B.: Bayesian inference for a discretely observed stochastic kinetic model. Stat. Comput. **18**(2), 125–135 (2008)

Calderhead, B., Girolami, M.: Estimating Bayes factors via thermodynamic integration and population MCMC. Comput. Stat. Data Anal. **53**(12), 4028–4045 (2009)

Del Moral, P., Penev, S.: Stochastic Processes: From Applications to Theory. CRC Press, Boca Raton (2017)

Del Moral, P., Doucet, A., Jasra, A.: Sequential Monte Carlo samplers. J. R. Stat. Soc. Ser. B **68**(3), 411–436 (2006)

Dupuis, P., Liu, Y., Plattner, N., Doll, J.D.: On the infinite swapping limit for parallel tempering. Multiscale Model. Simul. **10**(3), 986–1022 (2012)

Earl, D.J., Deem, M.W.: Optimal allocation of replicas to processors in parallel tempering simulations. J. Phys. Chem. B **108**(21), 6844–6849 (2004)

Fearnhead, P., Prangle, D.: Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation. J R. Stat. Soc. Ser. B **74**(3), 419–474 (2012)
Foreman-Mackey, D., Hogg, D.W., Lang, D., Goodman, J.: emcee: the MCMC hammer. Publ. Astron. Soc. Pac. **125**(925), 306 (2013)
Friel, N., Pettitt, A.N.: Marginal likelihood estimation via power posteriors. J. R. Stat. Soc. Ser. B **70**(3), 589–607 (2008)
Geyer, C.: Importance sampling, simulated tempering and umbrella sampling. *Handbook of Markov Chain Monte Carlo*, pages 295–311, (2011)
Geyer, C. J.: Markov chain Monte Carlo maximum likelihood. *Interface Foundation of North America*, (1991)
Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
Goodman, J., Weare, J.: Ensemble samplers with affine invariance. Commun. Appl. Math. Comput. Sci. **5**(1), 65–80 (2010)
Green, P.J.: Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. Biometrika **82**(4), 711–732 (1995)
Haario, H., Saksman, E., Tamminen, J., et al.: An adaptive Metropolis algorithm. Bernoulli **7**(2), 223–242 (2001)
Hoffman, M.D., Gelman, A.: The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res. **15**(1), 1593–1623 (2014)
Hritz, J., Oostenbrink, C.: Optimization of replica exchange molecular dynamics by fast mimicking. J. Chem. Phys. **127**(20), 204104 (2007)
Jasra, A., Stephens, D.A., Holmes, C.C.: Population-based reversible jump Markov chain Monte Carlo. Biometrika **94**(4), 787–807 (2007)
Karimi, K., Dickson, N., Hamze, F.: High-performance physics simulations using multi-core CPUs and GPGPUs in a volunteer computing context. Int. J. High Perform. Comput. Appl. **25**(1), 61–69 (2011)
Kone, A., Kofke, D.A.: Selection of temperature intervals for parallel-tempering simulations. J. Chem. Phys. **122**(20), 206101 (2005)
Lee, A.: On the choice of MCMC kernels for approximate Bayesian computation with SMC samplers. In *Simulation Conference (WSC), Proceedings of the 2012 Winter*, pages 1–12. IEEE, 2012
Lee, A., Łatuszyński, K.: Variance bounding and geometric ergodicity of Markov chain Monte Carlo kernels for approximate Bayesian computation. Biometrika **101**(3), 655–671 (2014)
Lingenheil, M., Denschlag, R., Mathias, G., Tavan, P.: Efficiency of exchange schemes in replica exchange. Chem. Phys. Lett. **478**(1–3), 80–84 (2009)
Lotka, A.J.: Elements of physical biology. Sci. Prog. Twent. Century 1919–1933 **21**(82), 341–343 (1926)
Marin, J.-M., Pudlo, P., Robert, C. P., Ryder, R. J.: Approximate Bayesian computational methods. *Stat. Comput.*, pages 1–14, (2012)
MATLAB. *version 9.7.0.1190202 (R2019b)*. The MathWorks Inc., Natick, Massachusetts, (2019)
Miasojedow, B., Moulines, E., Vihola, M.: An adaptive parallel tempering algorithm. J. Comput. Graph. Stat. **22**(3), 649–664 (2013)
Murray, I., Adams, R., and MacKay, D.: Elliptical slice sampling. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 541–548. JMLR Workshop and Conference Proceedings, (2010)
Murray, L. M., Singh, S., Jacob, P. E., and Lee, A.: Anytime Monte Carlo. *arXiv preprint* arXiv:1612.03319, (2016).
Nishihara, R., Murray, I., Adams, R.P.: Parallel MCMC with generalized elliptical slice sampling. J. Mach. Learn. Res. **15**(1), 2087–2112 (2014)
Prangle, D., et al.: Adapting the ABC distance function. Bayesian Anal. **12**(1), 289–309 (2017)
Pritchard, J.K., Seielstad, M.T., Perez-Lezaun, A., Feldman, M.W.: Population growth of human Y chromosomes: a study of Y chromosome microsatellites. Mol. Biol. Evol. **16**(12), 1791–1798 (1999)
Rathore, N., Chopra, M., de Pablo, J.J.: Optimal allocation of replicas in parallel tempering simulations. J. Chem. Phys. **122**(2), 024111 (2005)
Robert, C., Casella, G.: Monte Carlo Statistical Methods, chapter The Metropolis-Hastings Algorithm. Springer Texts in Statistics, Springer, New York (2004)978-1-4757-4145-2. https://doi.org/10.1007/978-1-4757-4145-2_7
Roberts, G.O., Rosenthal, J.S., et al.: Optimal scaling for various Metropolis-Hastings algorithms. Stat. Sci. **16**(4), 351–367 (2001)
Rodinger, T., Howell, P.L., Pomès, R.: Distributed replica sampling. J. Chem. Theory Comput. **2**(3), 725–731 (2006)
Sokal, A.: Monte Carlo methods in statistical mechanics: foundations and new algorithms. In *Functional integration*, pages 131–192. Springer, (1997)
Swendsen, R.H., Wang, J.-S.: Replica Monte Carlo simulation of spin-glasses. Phys. Rev. Lett. **57**(21), 2607 (1986)
Syed, S., Bouchard-Côté, A., Deligiannidis, G., Doucet, A.: Non-reversible parallel tempering: a scalable highly parallel MCMC scheme. *arXiv preprint* arXiv:1905.02939 (2019)
Tavaré, S., Balding, D.J., Griffiths, R.C., Donnelly, P.: Inferring coalescence times from DNA sequence data. Genetics **145**(2), 505–518 (1997)
Toni, T., Welch, D., Strelkowa, N., Ipsen, A., Stumpf, M.P.: Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. J. R. Soc. Interface **6**(31), 187–202 (2009)
Volterra, V.: *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari, (1927)
Wang, F., Jordan, K.: Parallel-tempering Monte Carlo simulations of the finite temperature behavior of $(H_2O)_6^-$. J. Chem. Phys. **119**(22), 11645–11653 (2003)
Wilkinson, D.J.: Stochastic Modelling for Systems Biology. CRC Press, Boca Raton (2011)